

Fundamentos de programación

con Python 3



Facebook:

<https://www.facebook.com/dogr4mcode.web>

[Jorge Nolasco Valenzuela
Javler Gamboa Cruzado
Luz Nolasco Valenzuela]



Dogram Code

Únete al grupo:

<https://www.facebook.com/groups/librosyrecursosdeprogramacion>



Perú - México - Colombia - Chile - Ecuador - España - Bolivia - Uruguay - Guatemala - Costa Rica

<https://dogramcode.com/programacion>



Fundamentos de programación con Python 3

Autores: Mag. Jorge Santiago Nolasco Valenzuela
Dr. Javier Arturo Gamboa Cruzado
Mag. Luz Elena Nolasco Valenzuela

© Derechos de autor registrados:
Empresa Editora Macro EIRL

© Derechos de edición, arte gráfico y diagramación reservados:
Empresa Editora Macro EIRL

Coordinación de edición:
Lucero Monzón Morán

Diseño de portada:
Fernando Cavassa Repetto

Corrección de estilo:
Karen Huachaca Avendaño

Diagramación:
Lucero Monzón Morán

Edición a cargo de:
© Empresa Editora Macro EIRL
Av. Paseo de la República N.º 5613, Miraflores, Lima, Perú

📞 Teléfono: (511) 748 0560
✉️ E-mail: proyectoeditorial@editorialmacro.com
🌐 Página web: www.editorialmacro.com

Primera edición digital: agosto de 2020

ISBN digital N.º 978-612-304-684-2

Prohibida la reproducción parcial o total, por cualquier medio o método, de este libro sin previa autorización de la Empresa Editora Macro EIRL.

JORGE NOLASCO VALENZUELA

Doctorando en Administración de Empresas con mención en Dirección Estratégica por la Universidad San Ignacio de Loyola. Cuenta con grado de magíster en Gestión de Tecnología de Información e ingeniero de Sistemas y Computación por la Universidad Inca Garcilaso de la Vega. Además, posee conocimientos y dominio de las TIC y de las herramientas Open Source. Es autor de los siguientes libros: *Desarrollo aplicaciones con Android*, *Python Aplicaciones prácticas*, *Java y Android Studio y Android y J2ME*.

En la actualidad, ejerce la docencia en diversas universidades. Entre sus principales ámbitos de investigación se encuentran la informática y las nuevas tecnologías. Colabora frecuentemente en proyectos empresariales en organizaciones relacionadas al mercado de valores, mineras y constructoras.

JAVIER ARTURO GAMBOA CRUZADO

Profesor Investigador RENACYT-Nivel I. Es doctor en Administración de Empresas por la Universidad Nacional Mayor de San Marcos, doctor en Ingeniería de Sistemas por la Universidad Nacional Federico Villarreal, magíster en Gestión de la Información y del Conocimiento por la Universidad Montpellier III Francia-Universidad Nacional Mayor de San Marcos, magíster en Ingeniería de Sistemas por la Universidad Nacional de Ingeniería (UNI) e ingeniero de sistemas. Además, tiene un diplomado en Business Analytics (Certificación Green Belt-Six Sigma por la North Carolina State University). Es especialista en Ciencia de Datos.

Cuenta con experiencia en la cátedra universitaria a nivel de pregrado y posgrado en distintas universidades de Lima y provincias. Ha realizado publicaciones de artículos científicos en revistas indexadas de gran prestigio.

A nivel profesional, ha desempeñado diversos cargos empresariales y realiza múltiples consultorías a empresas públicas y privadas de distintos sectores empresariales. Actualmente es docente en la Universidad Nacional Mayor de San Marcos y consultor asociado en Heedcom del Perú. Business Analytics Consultant en Business and Technology Services VIP, LLC (EE. UU.) desde el 2012 en los ámbitos de Data Science (Business Intelligence, Business Analytics, Web Mining, Text Analytics y Big Data), así como de la mejora de procesos (BPM y Six Sigma).

LUZ ELENA NOLASCO VALENZUELA

Docente universitaria y magíster en Derecho otorgado por York University, Osgoode Hall Law School en Toronto (Canadá). Actualmente, cursa estudios de maestría en Educación con mención en Informática y Tecnología Educativa. Graduada con honores como licenciada en Humanidades en las especialidades de Criminología, Derecho y Sociedad otorgada por York University. Además, es licenciada en Educación y Ciencias Humanas, Educación Secundaria con especialidades en Matemática y Física por la Universidad Nacional Federico Villarreal. Actualmente, es candidata en la segunda maestría en Derecho Civil y Comercial por la Universidad Nacional Federico Villarreal.

Cuenta con más de diez años de experiencia en la docencia a nivel superior y secundario en diversas instituciones educativas a nivel nacional como internacional. Tiene dominio a nivel nativo del idioma inglés y español, y conocimientos del idioma italiano.

Sus áreas de investigación están focalizadas en la educación y en el derecho. En el ámbito educativo, está interesada en los temas de planeamiento estratégico de tecnológicas de información, en cómo aplicar las tecnologías en diversos contextos, así como en la intersección entre las matemáticas y las tecnologías y la aplicación de las tecnologías en el aprendizaje de las matemáticas. En el ámbito jurídico, se interesa en temas como el arbitraje internacional, el derecho internacional privado, la minería a nivel global, el derecho laboral, el derecho empresarial y la cultural y la sociedad.



*Dedicado a quienes no cesan en su
afán de su superación y cambio. El
cambio, lo único constante.*

Índice

Presentación	13
1 Introducción a Python	17
1.1 Instalación de Python en Linux	17
1.2 Instalación de Python en Windows	17
1.3 Comprobar la instalación	19
1.4 Realizar pruebas	20
1.5 Operaciones matemáticas	20
1.6 Entornos de trabajo- PyCharm	21
1.7 Código legible	27
1.8 Crear el proyecto Hola Mundo	27
1.9 Función print	31
1.9.1 Efectos de la función print	32
1.9.2 La función print: usando múltiples argumentos	33
1.9.3 La función print: palabras claves	33
1.9.4 La función print: usando caracteres de escape	34
1.10 Crear el proyecto Formato	34
1.11 Variables	39
1.12 Constantes	40
1.13 Tipos básicos	40
1.14 Múltiple asignación	43
1.15 Función type	43
1.16 Conversión de datos	44

1.17 Resolución de problemas matemáticos	45
1.18 Otros tipos	46
1.19 Comentarios	47
1.20 Operaciones con cadena	47
1.20.1 Concatenación de cadenas	47
1.20.2 Multiplicar una cadena por un número	48
1.20.3 Longitud de una cadena	48
1.20.4 Manejo de segmentos de una cadena	50
1.20.5 Operador in	51
1.20.6 Convertir mayúsculas, minúsculas y otros	51
1.20.7 División en trozos	52
1.20.8 Formatos de cadenas - str.format()	53
1.20.9 Convertir números a cadena - str()	54
1.20.10 UTF-8: codificación de caracteres	56
1.21 La clase math	56
1.22 Generación de números aleatorios	57
1.23 Fechas y horas	58
1.23.1 Ejemplos	59
1.24 Más sobre comentarios	64
1.25 Operadores	65
1.26 Algo más sobre operadores	66
1.27 Operadores de acceso directo	67
1.28 Concatenación	67
1.29 Replicando	68
1.30 Operador ==	69
1.31 Operador !=	70
1.32 Desplazamiento de bits	70
1.33 Tabla de prioridad de operadores	71
Preguntas: Capítulo 1	72

2 Estructura de control..... 75

2.1 Instrucciones de control.....	75
2.1.1 If	75
2.1.2 While	79
2.1.3 While y else	81
2.1.4 For	81
2.1.5 For y else.....	84
2.2 Entrada y salida estándar.....	84
2.2.1 Más sobre la función print.....	84
2.2.2 Cadenas f.....	86
2.2.3 Break y continue.....	86
2.2.4 Input	87
2.2.5 Tipos de casting.....	89
2.2.6 Conversión a cadena	89
2.3 Funciones	90
2.3.1 Crear el proyecto Funciones.....	91
2.3.2 Ejemplos	97
2.3.3 Funciones con parámetros no definidos	101
2.3.4 Funciones recursivas	102
2.4 Módulos y paquetes.....	104
Preguntas: Capítulo 2	113

3 Listas, tuplas, diccionarios, conjuntos y excepciones 117

3.1 Listas	117
3.1.1 Posición de elementos	118
3.1.2 La función Len	118
3.1.3 Índices negativos.....	120
3.1.4 Cambiar elementos a una lista	120
3.1.5 Añadir elementos a una lista	121
3.1.6 Eliminar elementos a una lista	121

3.1.7 Algunos ejemplos de listas.....	122
3.1.8 Intercambiar elementos de una lista.....	123
3.1.9 Ordenamiento de listas.....	123
3.1.10 Método sort().....	124
3.1.11 Método reverse().....	125
3.1.12 Limitar los elementos de una lista.....	125
3.1.13 Listas en listas.....	126
3.1.14 Listas bidimensionales.....	128
3.2 Tuplas	129
3.2.1 Ejemplos.....	129
3.2.2 Creación de tuplas vacías.....	130
3.2.3 Mostrar elementos de una tupla.....	130
3.3 Diccionarios	131
3.3.1 Métodos de los diccionarios	132
3.3.2 Ejemplos.....	133
3.3.3 Función sorted().....	133
3.3.4 Utilizar los métodos item() y values().....	134
3.3.5 Modificar, ingresar y eliminar valores de un diccionario.....	135
3.3.6 Ingresar elementos a un diccionario	135
3.4 Conjuntos	136
3.4.1 Ejemplos.....	137
3.5 Excepciones.....	139
3.5.1 Algunos ejemplos de uso de excepciones.....	140
Preguntas: Capítulo 3	144

4 Programación orientada a objetos y sus funciones147

4.1 Programación orientada a objetos	147
4.1.1 Introducción a la POO	147
4.1.2 Definición de clase	147
4.1.3 Definición de objetos.....	149
4.1.4 Herencia.....	149

4.1.5 Nuestra primera clase	149
4.1.6 Explicación del código contador.py	154
4.1.7 Definición de clases- constructor	155
4.1.8 Ejemplos	158
4.1.9 Atributo __dict__	161
4.1.10 Herencia simple	162
4.1.11 Herencia múltiple	168
4.1.12 Conocer un objeto de una clase hija	169
4.1.13 Conocer un objeto de una clase específica	170
4.1.14 Otros ejemplos de uso de herencia	170
4.1.15 Iteradores y generadores	171
4.1.16 Ejemplo de métodos especiales	172
4.1.17 Polimorfismo	173
5 Manejo de ficheros	175
5.1 Manejo de ficheros	175
5.1.1 Ejercicios de archivos TXT	176
5.1.2 Ejercicios de archivos binarios	179
5.1.3 Archivos JSON	182
5.2 Expresiones regulares	183
5.2.1 Ejemplos	185
Preguntas: Capítulo 5	189
Referencias bibliográficas	191
Referencias electrónicas	191

Presentación

Python es un lenguaje de programación muy popular de propósito general creado en los 90 por Guido van Rossum, quien trabajó en Google y, en la actualidad, se desempeña en Dropbox.

El creador de Python nombró así a este lenguaje en honor al cómic y programa de televisión Monty Python.

Posee una amplia comunidad de desarrolladores que giran en torno a él y buscan aportar, compartir y construir software escalable en comunidad.

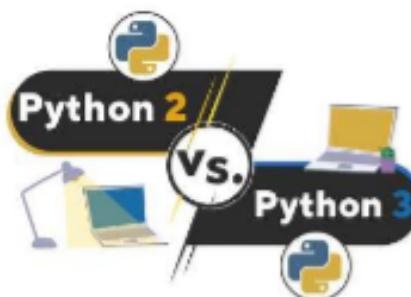
Tiene un ecosistema muy amplio que atrae a programadores, investigadores y profesionales de otras áreas que buscan mejorar su rendimiento laboral haciendo uso del código.

Asimismo, posee una sintaxis muy limpia y legible. Cuenta con tipado dinámico, es decir, que una variable puede poseer datos de varios tipos. Además, por su naturaleza interpretada, este lenguaje es fácil de aprender. Python es un lenguaje interpretado. Por ello, no se requiere compilar el código fuente para poder ejecutarlo. Esto brinda ventajas como la rapidez de desarrollo.



Python está escrito en el lenguaje C. Debido a ello, se puede extender a través de su API en C o C++ y escribir nuevos tipos de datos, funciones, etc.

Actualmente, hay dos vertientes: la versión 2.x y la 3.x. Es posible que en algún momento ambas versiones se integren. Es recomendable utilizar la última versión estable 3.x.



Una de las características más importantes de Python es que es multiparadigma: programación estructurada, programación orientada a objetos y programación funcional.

En el desarrollo web de Python, se pueden utilizar los framework: Django y Flask. Entre las empresas más conocidas que utilizan Python figuran la NASA, Dropbox e Instagram.

Existen otros proyectos realizados con Python:

- Pinterest
- Battlefield 2
- BitTorrent
- Ubuntu Software Center
- Panda 3
- Google App Engine

Se encuentra información más detallada de otros proyectos con Python en el siguiente sitio web:
<https://www.escuelapython.com/grandes-proyectos-hechos-python/>

En Data Science y Machine Learning se cuenta con Pandas, Scikit-Learn y TensorFlow. Además, Python es multiplataforma: Linux, Windows, Mac OS, Solaris, etc.

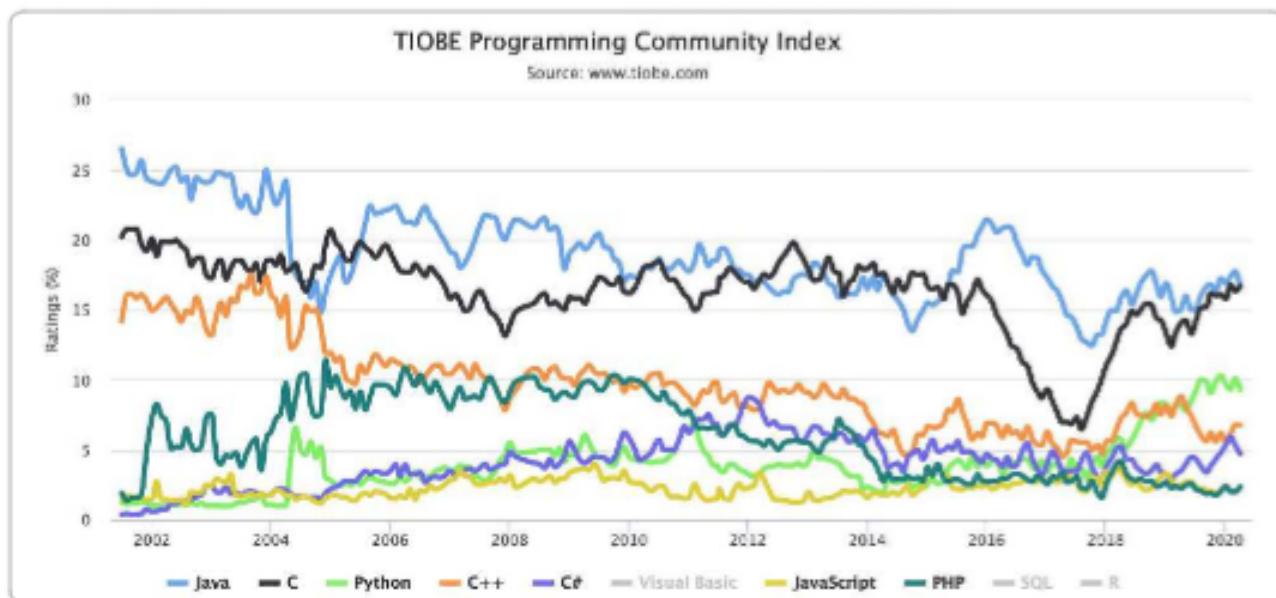
Con Python se puede crear desde un sitio web hasta un programa o aplicación para efectuar cierta tarea científica como calcular valores estadísticos o resolver matemática compleja. Además, se puede desarrollar lo siguiente:

- Juegos
- Desarrollo web
- Gráficos y diseño
- Aplicaciones financieras
- Ciencia
- Automatización de diseño electrónico
- Desarrollo de software
- Software dedicado a negocios

Asimismo, Python permite ser implementado en diferentes lenguajes:

- CPython - Python tradicional escrito en C
- Jython - Python para la JVM
- IronPython - Python para .NET
- Pypy - Python más rápido con compilador JIT
- StacklessPython - Branch de CPython con soporte para microthreads

En la actualidad, Python es uno de los lenguajes de programación más populares y extendidos, así lo demuestra el índice TIOBE.



Rank	Apr 2020	Apr 2019	Change	Programming Language	Ratings	Change
1	1	1		Java	16.73%	+1.69%
2	2	2		C	16.72%	+2.64%
3	4	4	▲	Python	9.31%	+1.15%
4	3	3	▼	C++	6.78%	-2.06%
5	6	6	▲	C#	4.74%	+1.23%
6	5	5	▼	Visual Basic	4.72%	-1.07%
7	7	7		JavaScript	2.38%	-0.12%
8	9	9	▲	PHP	2.37%	+0.13%
9	8	8	▼	SQL	2.17%	-0.10%
10	16	16	▲	R	1.54%	+0.35%

Fuente: <https://www.tiobe.com/tiobe-index/>

Por otro lado, el índice PYPL (PopularitY of Programming Language) considera también a Python como el lenguaje más popular y el que ha experimentado un mayor crecimiento en estos últimos años.

PYPL PopularitY of Programming Language				
Worldwide, Aug 2020 compared to a year ago.				
Rank	Change	Language	Share	Trend
1		Python	31.59 %	+3.3 %
2		Java	16.9 %	-2.7 %
3		JavaScript	8.17 %	+0.0 %
4		C#	8.54 %	-0.7 %
5	↑	C/C++	5.88 %	+0.1 %
6	↓	PHP	5.28 %	-0.7 %
7		R	4.18 %	+0.3 %
8		Objective-C	2.8 %	-0.0 %
9		Swift	2.35 %	-0.0 %

The PYPL PopularityY of Programming Language Index is created by analyzing how often language tutorials are searched on Google.

Fuente: <http://pypl.github.io/PYPL.html>

A continuación, estas son las fortalezas de Python:

- Fácil de aprender
- Fácil de enseñar
- Fácil de usar
- Fácil de entender
- Fácil de obtener, instalar e implementar

Este libro te permitirá conocer de manera fácil los conceptos y características básicas del lenguaje de programación Python; es decir, podrás interpretarlo y experimentar su uso.

1

Introducción a Python

1.1 Instalación de Python en Linux

Python viene instalado de forma predeterminada en los sistemas operativos Mac y Linux, ya que es utilizado intensamente por muchos componentes del sistema operativo Linux.

Si es usuario de Linux, abra la terminal/consola y escriba:



```
python3
```

En el indicador de shell, presione <Enter> y espere. Observará lo siguiente:

```
Python 3.4.5 (default, Jan 12 2017, 02:28:40)
[GCC 4.2.1 Compatible Clang 3.7.1 (tags/RELEASE_37-l-branch)]
Type "help", "copyright", "credits" or "license"
>>>
```

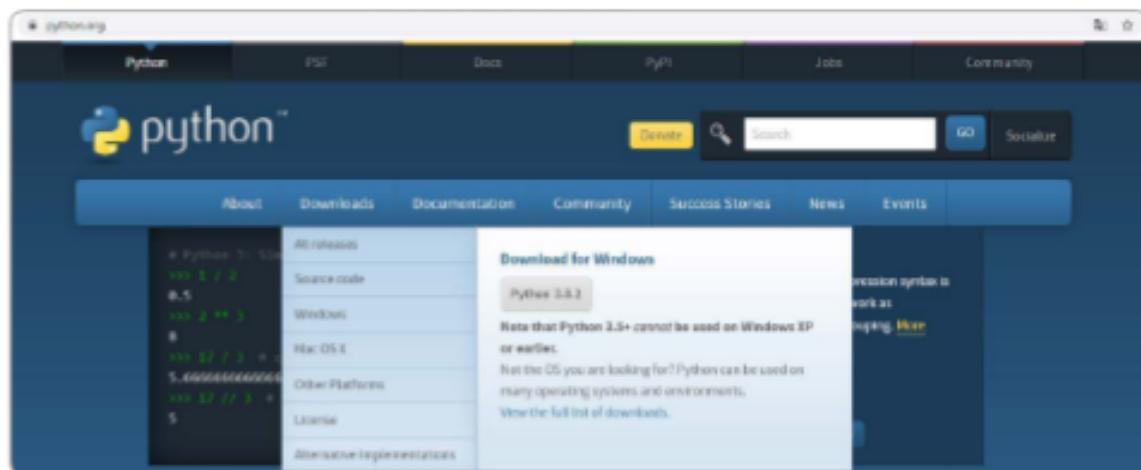
1.2 Instalación de Python en Windows

Si es usuario de Windows, puede instalar la versión más reciente para asegurarse de tener las últimas actualizaciones.



Paso 1:

Ingrese a la siguiente URL para descargar: <https://www.python.org/>.



The screenshot shows the Python.org homepage. The navigation bar includes links for Python, PSF, Docs, PyPI, Jobs, and Community. The main content area features the Python logo and a search bar. A sidebar on the left lists Python versions from 3.1 to 3.6. The central area displays a "Download for Windows" section for Python 3.6.3, with a note that Python 3.6+ cannot be used on Windows XP or earlier. It also mentions that Python can be used on many operating systems and environments, with a link to view the full list of downloads.



Puede descargar directamente la última versión desde la siguiente URL:
<https://www.python.org/downloads/>.

Paso 2:

Ejecute el instalador descargado.

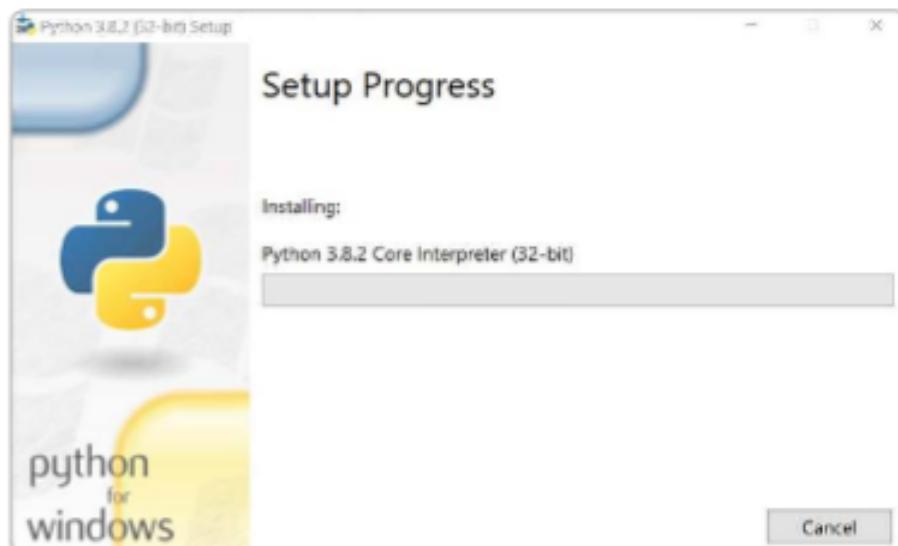
**Paso 3:**

Inicie el proceso de instalación y haga clic en **Install Now**.

**Nota**

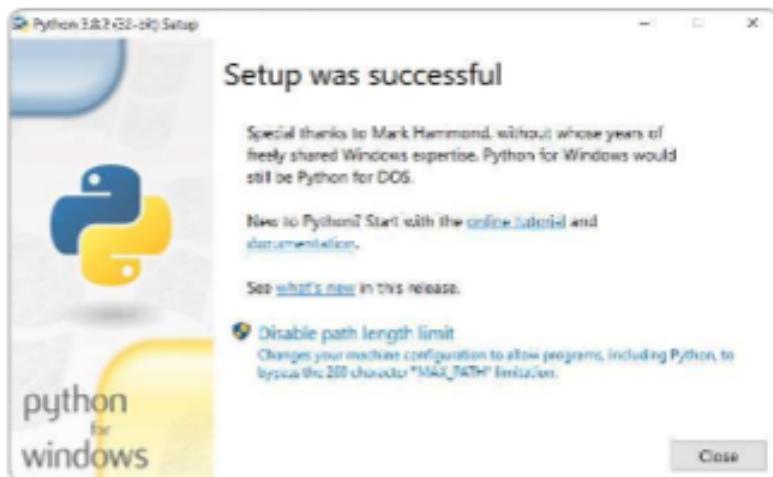
También puede hacer clic en **Add Python 3.8 to PATH** para poder ejecutar programas desde la línea de comandos.

A continuación, se iniciará el proceso de instalación. Esto puede durar algunos minutos.



Paso 4:

Una vez completada la instalación, se mostrará una pantalla, donde debe presionar el botón **Close** para cerrar el programa de instalación.

**Nota**

Es recomendable tener desactivada la opción **Disable path length limit**. Si alguna aplicación necesita tenerla activada, se deberá modificar la siguiente clave:

`HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem\LongPathsEnabled`

El valor 0 significa que el límite está desactivado y el valor 1 significa que está activado.

1.3 Comprobar la instalación

Para el sistema operativo Windows, abra símbolos del sistema e ingrese el comando Python, como se muestra en la imagen de abajo. El intérprete interactivo debe abrirse. Si se desea salir, escriba `quit()`.

The screenshot shows a Windows Command Prompt window titled "Símbolo del sistema". It displays the following text:
 Microsoft Windows [Versión 10.0.18363.778]
 (c) 2019 Microsoft Corporation. Todos los derechos reservados.
 C:\Users\User>python
 Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32
 bit (Intel)] on win32
 type "help", "copyright", "credits" or "license" for more information.
 >>> quit()
 C:\Users\User>

Nota

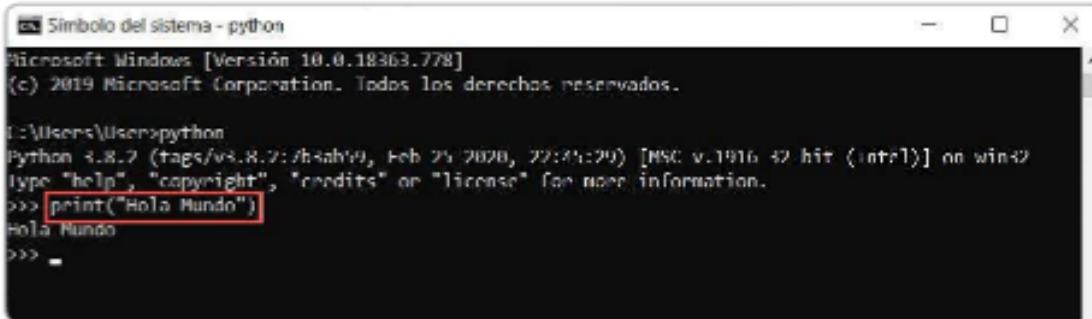
En Windows acceda a través de `<Win + R>` y escriba **CMD**. Luego haga clic en **Aceptar**.

Para el sistema operativo Mac o Linux, abra símbolos del sistema e ingrese el comando <python3>. Si desea salir, escriba **quit()**.

```
$ python3
Python 3.5.2 (default, Oct 3 2017, 17:46:00)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
$
```

1.4 Realizar pruebas

Los tres corchetes angulares (>>>) indican que usted está en el intérprete interactivo de Python y puede ya realizar su primera prueba.



```
Microsoft Windows [Versión 10.0.18363.778]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

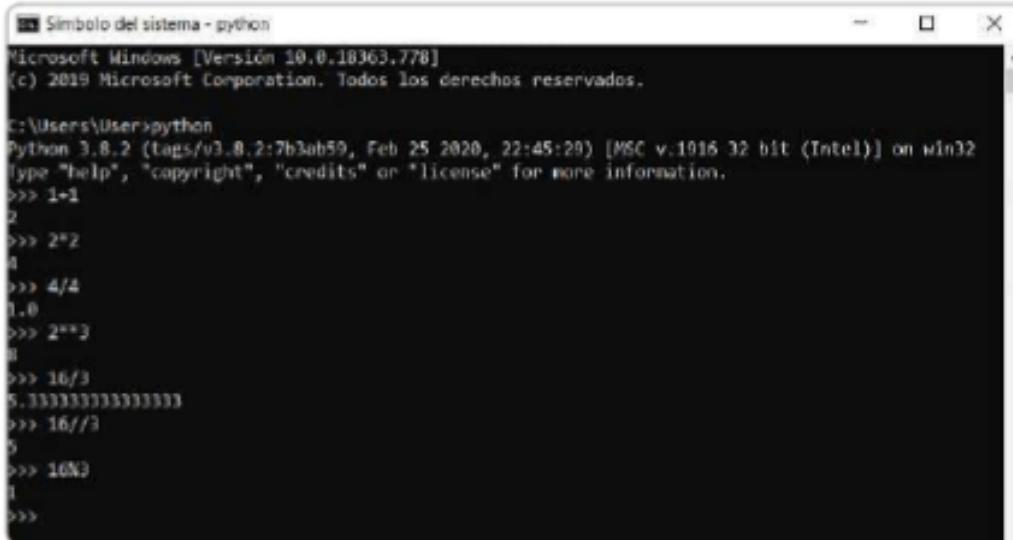
C:\Users\User>python
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hola Mundo")
Hola Mundo
>>>
```

Nota

La sesión del intérprete de Python es llamada también Python shell.

1.5 Operaciones matemáticas

A continuación, se muestra la sintaxis de Python para usar con los operadores matemáticos más comunes.



```
Microsoft Windows [Versión 10.0.18363.778]
(c) 2019 Microsoft Corporation. Todos los derechos reservados.

C:\Users\User>python
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 1+1
2
>>> 2^2
4
>>> 4/4
1.0
>>> 2**3
8
>>> 16/3
5.333333333333333
>>> 16//3
5
>>> 16%3
1
>>>
```

1.6 Entornos de trabajo-PyCharm

PyCharm es un entorno de desarrollo integrado (IDE) utilizado específicamente para el lenguaje Python. Para descargarlo, ingrese a la siguiente URL: <https://www.jetbrains.com/pycharm/>

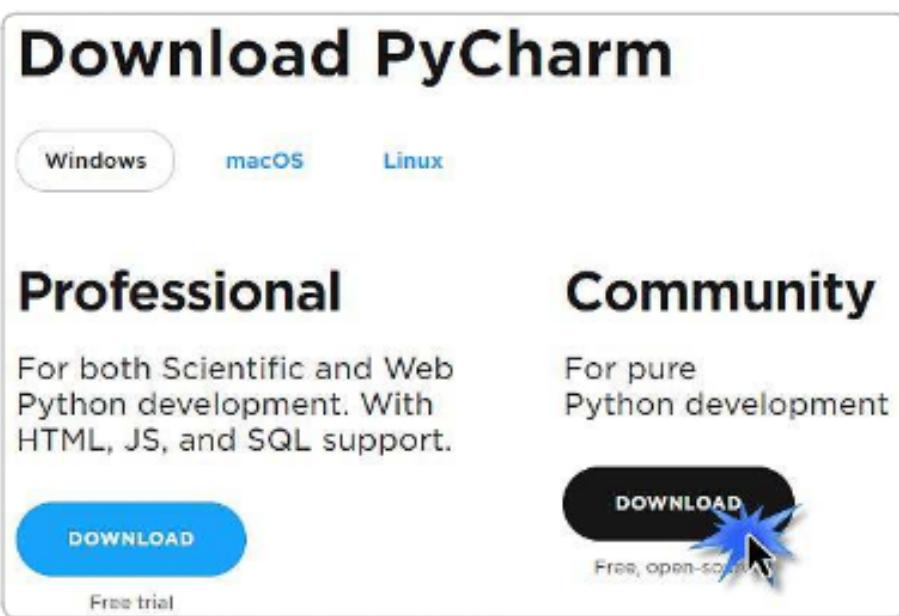
Paso 1:

Presione en el botón **DOWNLOAD**.



Paso 2:

Puede instalar la versión gratuita Pycharm Community, presionando en el botón **DOWNLOAD**.



Paso 3:

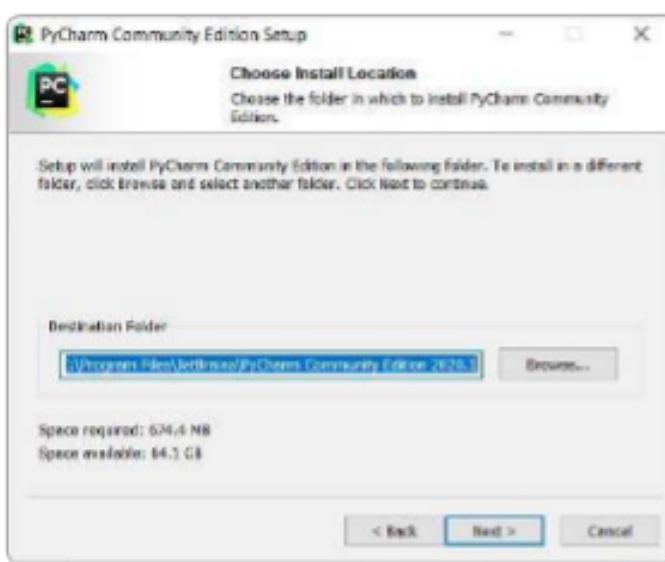
Una vez que se complete la descarga, ejecute el instalador.

**Paso 4:**

Haga clic en **Next**.

**Paso 5:**

Haga clic en **Next**.

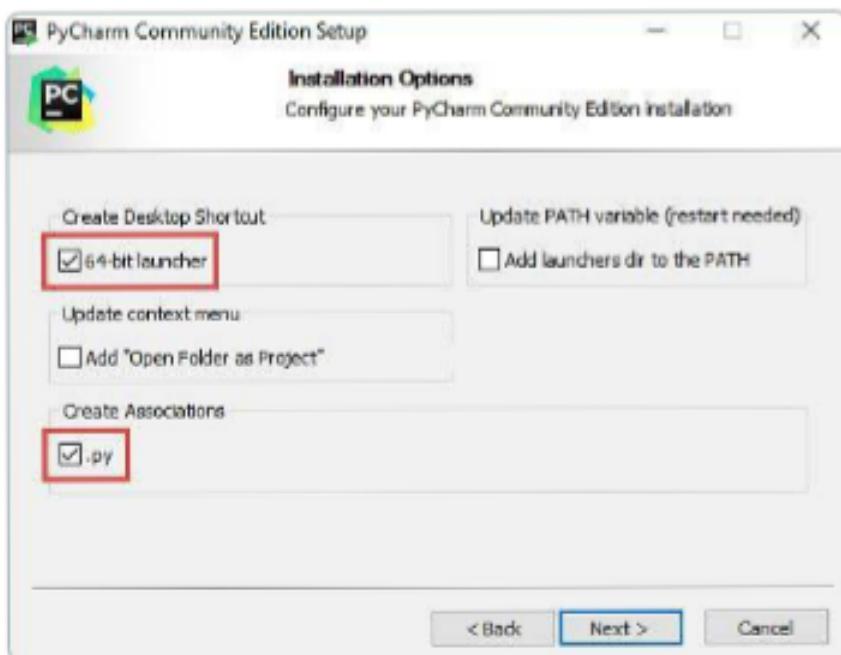
**Nota**

Puede cambiar la ruta de instalación, si fuera necesario.



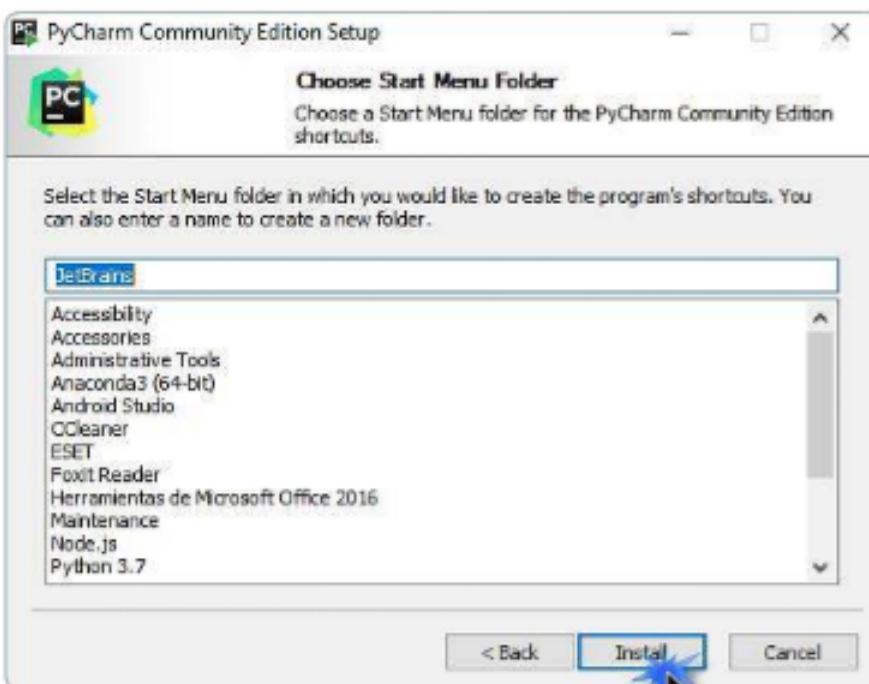
Paso 6:

Si desea crear un acceso directo de escritorio, haga clic en **Next**.



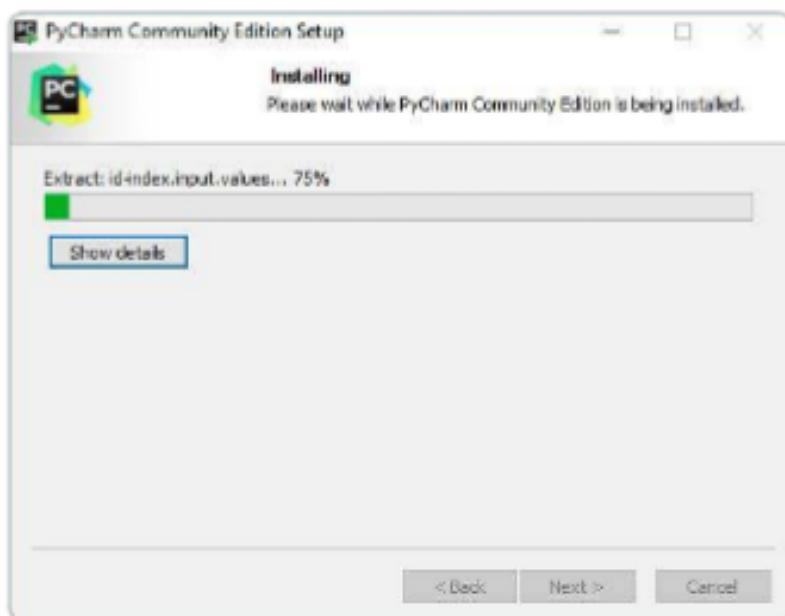
Paso 7:

Luego, haga clic en **Install**.



Paso 8:

Espere que termine la instalación.



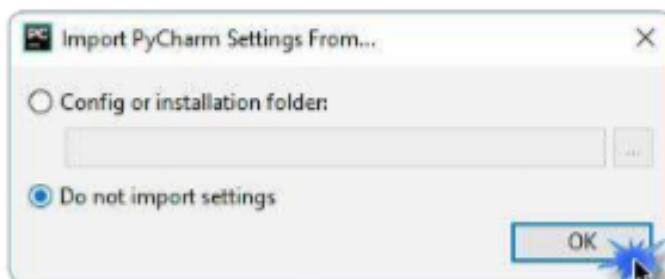
Paso 9:

Al finalizar la instalación, recibirá una pantalla de mensaje que PyCharm está instalado. Si desea continuar y ejecutarlo, haga clic en **Run PyCharm Community Edition**, luego en **Finish**.



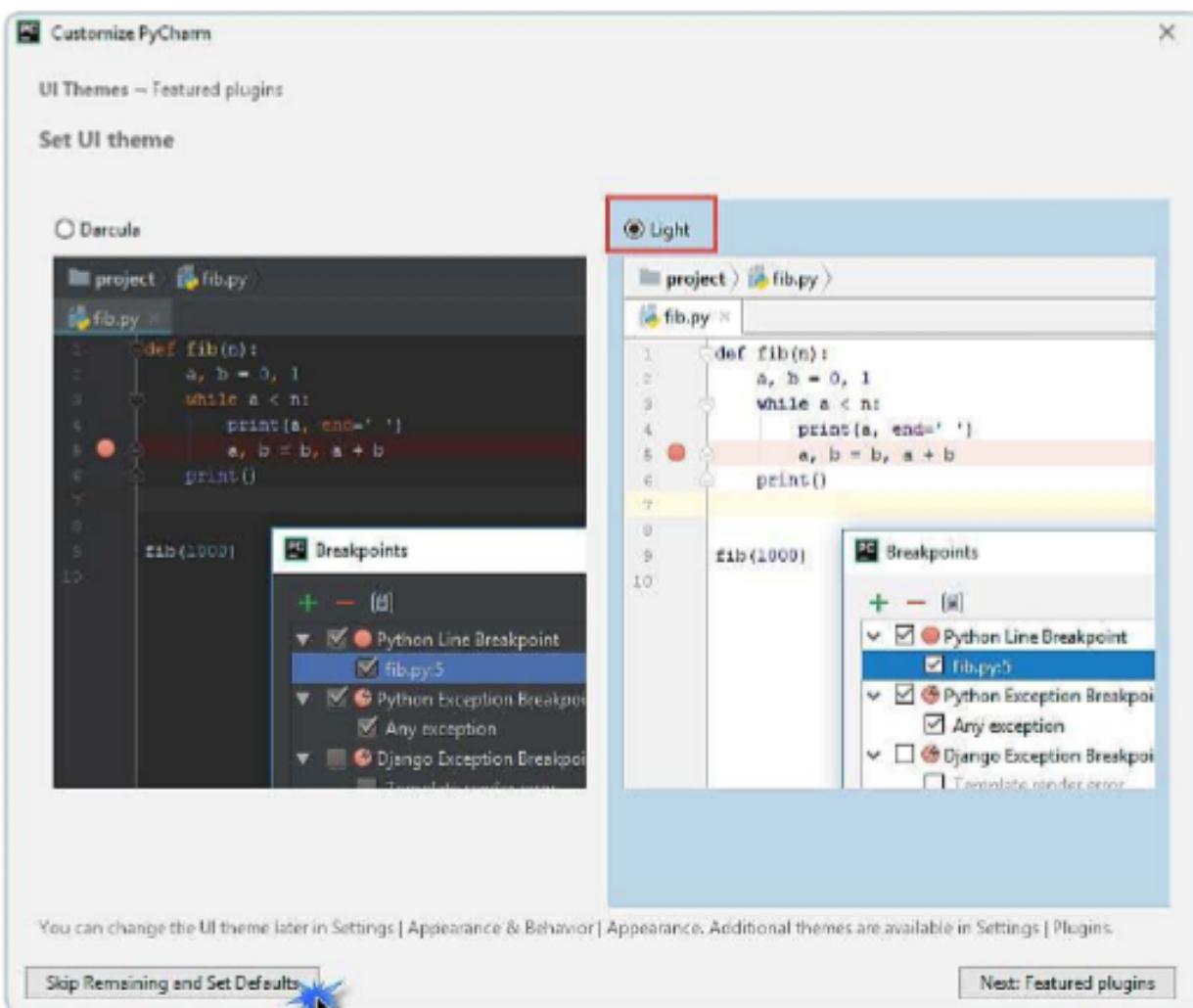
Paso 10:

Haga clic en **OK**.



Paso 11:

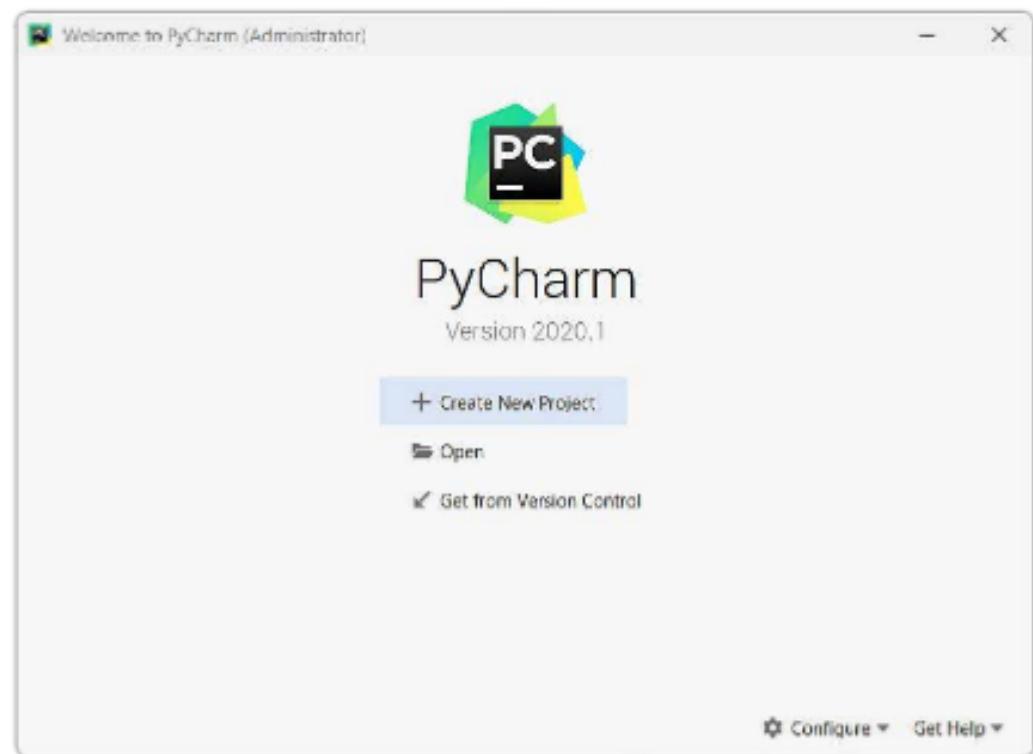
Seleccione el tema y haga click en **Skip Remaining and Set Defaults** para establecer las configuraciones predefinidas por el instalador.





Paso 12:

Después de hacer clic en **Finalizar**, aparecerá la siguiente pantalla:



1.7 Código legible

Para explicar de una manera general la sintaxis de Python, se escribirá el clásico Hola Mundo:



En Python se manejan las indentaciones o tabulaciones. Es suficiente utilizar 4 espacios. Así se demuestra al escribir la siguiente función:

```
def saludo():
    print("Hola Mundo")
```

A red box highlights the number "4" in the indentation of the first line, with a callout box below it labeled "4 espacios".

Nota


Existen los PEPs (Python Enhancement Proposals) que son las propuestas de mejoras de Python.

1.8 Crear el proyecto Hola Mundo

Ahora se procederá a crear el proyecto Hola Mundo.

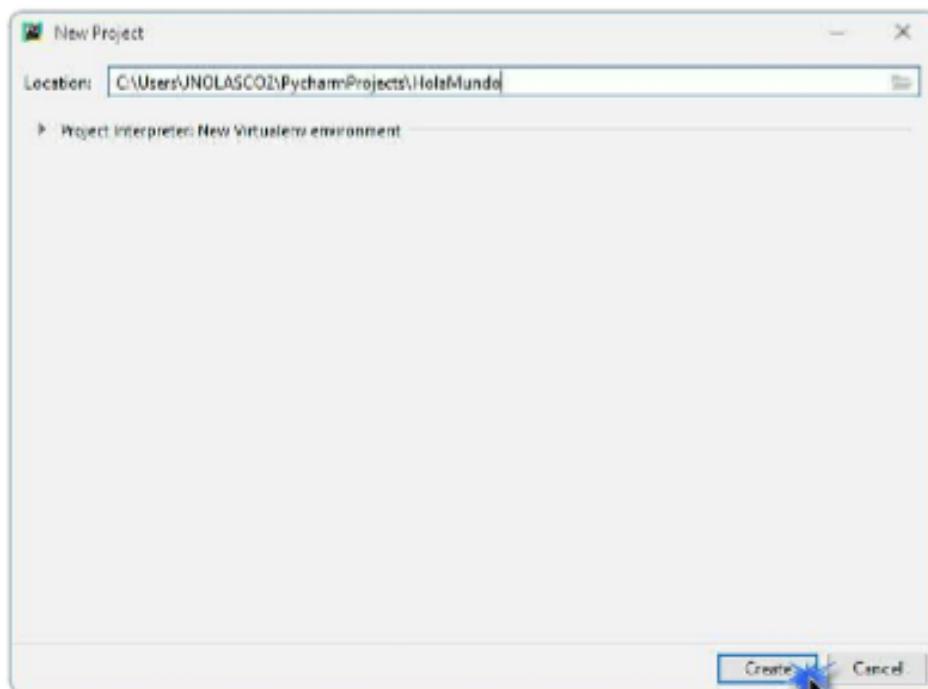
Paso 1:

Haga clic en **+ Create New Project**.



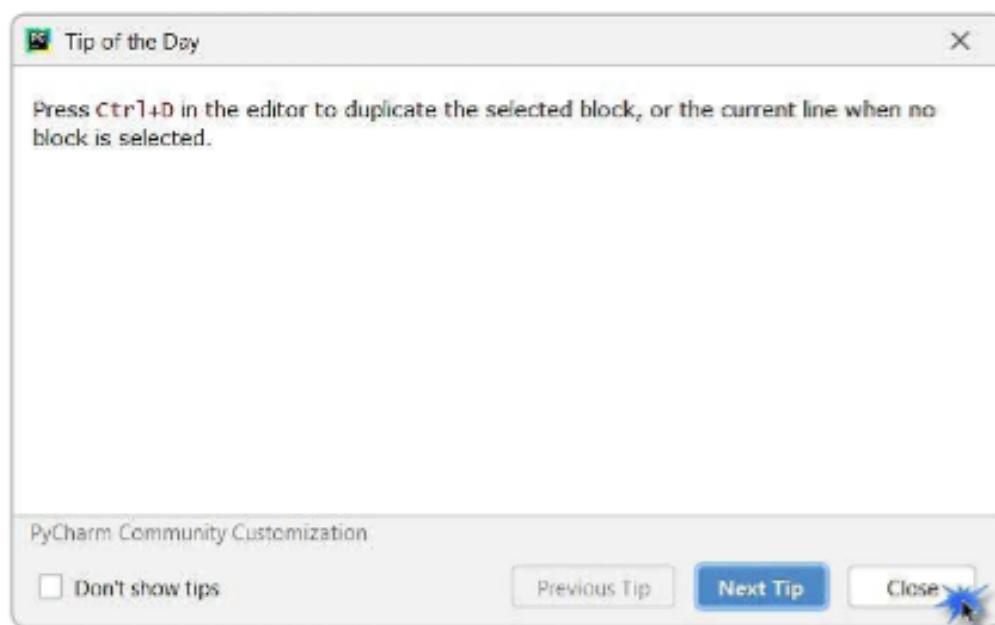
Paso 2:

Escriba junto el nombre del proyecto: **HolaMundo**. Luego, haga clic en **Create**.



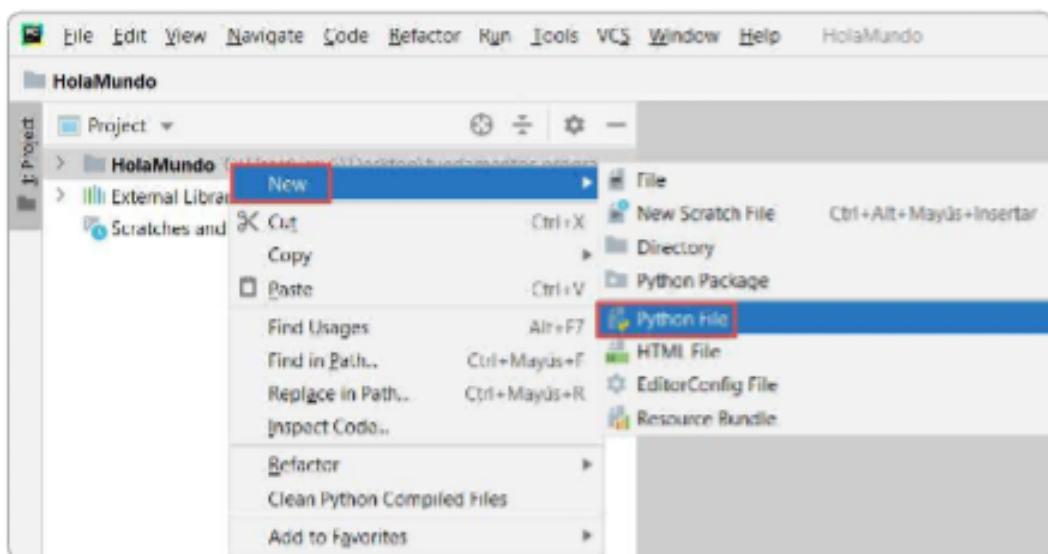
Paso 3:

Al finalizar, recibirá la bienvenida por parte de Pycharm. Luego, haga clic en **Next Tip**.



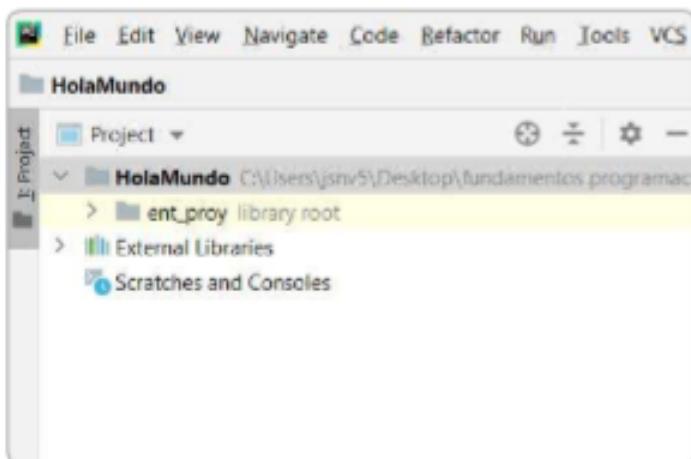
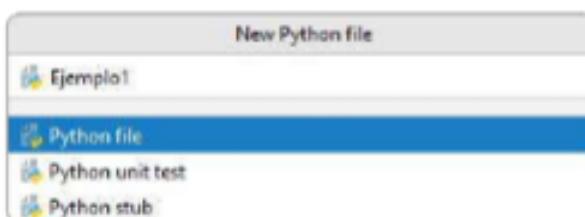
Paso 4:

Ahora debe crear un archivo nuevo. Para ello, haga clic derecho en la carpeta creada **HolaMundo**. Luego, en **New** y en **Python File**. Posteriormente, nombre el archivo.



Paso 5:

Escriba el nombre del archivo: **Ejemplo1**. Luego, presione <Enter>.



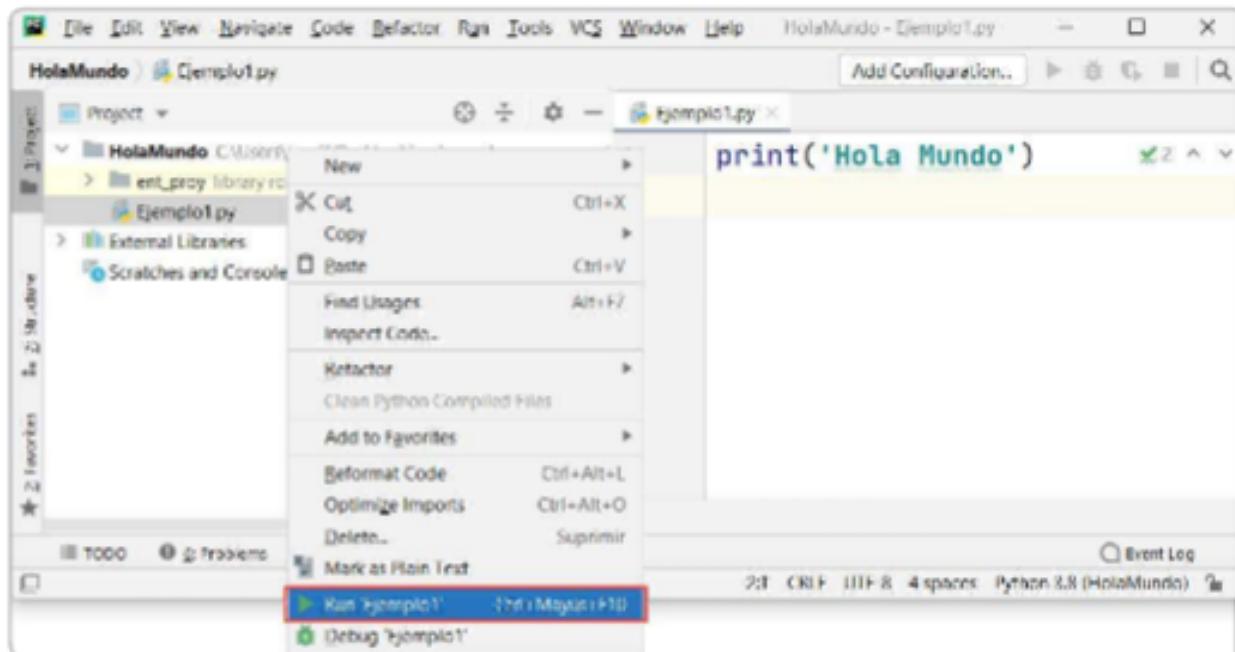
Paso 6:

Escriba el código y ejecute.

The screenshot shows the PyCharm IDE interface. The title bar says "HolaMundo - Ejemplo1.py". The left sidebar shows a project structure with "HolaMundo" selected, containing "ent_proy library root" and "Ejemplo1.py". The main editor window displays the Python code: `print('Hola Mundo')`. The status bar at the bottom indicates "2:1 CRLF UTF-8 4 spaces Python 3.8 (HolaMundo) 8e".

Paso 7:

Para ejecutarlo, haga clic derecho en el archivo **Ejemplo1.py**. Luego, seleccione **Run "Ejemplo1"**.



The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, Help, and the current project name HolaMundo - Ejemplo1.py. The title bar also displays the project name and file path. On the left, the Project tool window shows a tree structure with 'HolaMundo' as the root, containing 'ent_proy' (library root) and 'Ejemplo1.py'. Below that are External Libraries and Scratchpad and Console. The main editor window contains the Python code:

```
print('Hola Mundo')
```

. The bottom Run tool window shows the output: 'Hola Mundo' with a red box around it, followed by 'Process finished with exit code 0'.



La palabra "print" es el nombre de la función. Su significado proviene del contexto en el que es utilizada. En este caso, permite mostrar por consola Hola mundo.

1.9 Función print

La función print() se utiliza para mostrar información en la pantalla. A continuación, se muestra su sintaxis:

`print(valor1,valor2,...,valorn)`

Estos son algunos ejemplos:

Código	Salida estándar	Significado
print('Python')	Python	Muestra: Python
print(20)	20	Muestra: 20
nombre = "eva" print(nombre)	eva	Muestra el contenido de la variable nombre: eva
edad = 16 print(edad)	16	Muestra el contenido de la variable edad: 16
nombre = "eva" apellido = "perez" print(nombre, apellido)	eva perez	Muestra el contenido de las variables nombre y apellido: eva perez
nombre = "eva" apellido = "perez" co = nombre + " " + apellido print(co)	eva perez	Concatena el nombre y apellido, y muestra: eva perez
print()	Imprime una línea	Imprime una línea
print("edad:%2d"% (20))	20	Versiones antiguas
print("edad:{0:2d}".format(20))	20	Versiones nuevas

edad=29 sueldo=1200.161 print("edad:%2d sueldo:%8.3f"%(edad,sueldo))	edad: 29 sueldo:1200. 161	Versiónes antiguas
edad=29 sueldo=1200.161 print("edad:{0:2d} sueldo:{1:8.3f} ".format(edad,sueldo))	edad: 29 sueldo:1200. 161	Versiónes nuevas

Nota

Se cuenta con los siguientes formatos:

%8.3f

{1:8.3f}

% = flags

8 = ancho

3 = precisión (decimales)



Al utilizar la función print de la siguiente manera: print("Hola a Todos"), se generará el siguiente error:

SyntaxError: EOL while scanning string literal

1.9.1 Efectos de la función print

Los efectos son muy útiles, ya que esta función realiza las siguientes acciones:

- Toma argumentos. Puede aceptar más de un argumento o menos de uno.
- Convierte los argumentos en una forma legible para los seres humanos, si es necesario. Las cadenas no requieren esta acción, ya que ellas son legibles.
- Envía los datos resultantes al dispositivo de salida (generalmente la consola). Es decir, lo que coloque en la función print() aparecerá en su pantalla. Por ello, resulta muy útil esta función para ver los resultados de las operaciones y evaluaciones que realice.

Estos son los argumentos que espera la función print():

- Cadenas
- Números
- Caracteres
- Valores lógicos
- Objetos

Nota

La función print no retorna ningún valor.

1.9.2 La función print: usando múltiples argumentos

Se puede utilizar la función `print()` con más de un argumento. A continuación, se muestra un ejemplo:

```
print("Python", "es uno de los lenguajes" , "más utilizados.")
```

Al ejecutar el código, se observará la siguiente salida:

```
Python es uno de los lenguajes más utilizados.
```

1.9.3 La función print: palabras claves

Python ofrece otro mecanismo para pasar argumentos y modificar el comportamiento de la función `print()`. Este mecanismo se llama "argumentos de palabras claves", debido a que el significado de estos argumentos no se toma de su ubicación (posición), sino de la palabra especial "palabra clave" utilizada para identificarlos.

La función `print()` tiene dos argumentos de palabras claves que puede usar para sus propósitos. Uno de ellos se llama "end".

A continuación, se muestra un ejemplo:

```
print("Soy ", " Guido van Rossum ", end="")
print("Creador de Python.")
```

Al ejecutar el código, se observará la siguiente salida:

```
Soy Guido van Rossum creador de Python.
```



El argumento "end" determina los caracteres que la función `print()` envía a la salida una vez que alcanza el final de sus argumentos posicionales.

El comportamiento implícito es el siguiente: `end="\n"`.

También se puede indicar que el separador sea un carácter como (-). A continuación, se muestra un ejemplo:

```
print("Mi", "Nombre", "es", "Jorge", "Nolasco Valenzuela.", sep="-")
```

Al ejecutar el código, se observará la siguiente salida:

```
Mi-Nombre-es-Jorge-Nolasco Valenzuela.
```



La función `print()` ahora usa un guion, en lugar de un espacio, para separar los argumentos generados.

1.9.4 La función print: usando caracteres de escape

La barra invertida (\) tiene un significado importante cuando se usa dentro de las cadenas. Es llamado "el carácter de escape". Por la palabra "escape", debe entenderse que la serie de caracteres en la cadena se escapa, es decir, se detiene por un momento muy corto para introducir una inclusión especial.

En otras palabras, la barra invertida no significa nada, sino que es solo un tipo de anuncio, que indica que el siguiente carácter después de la barra invertida también tiene un significado diferente.

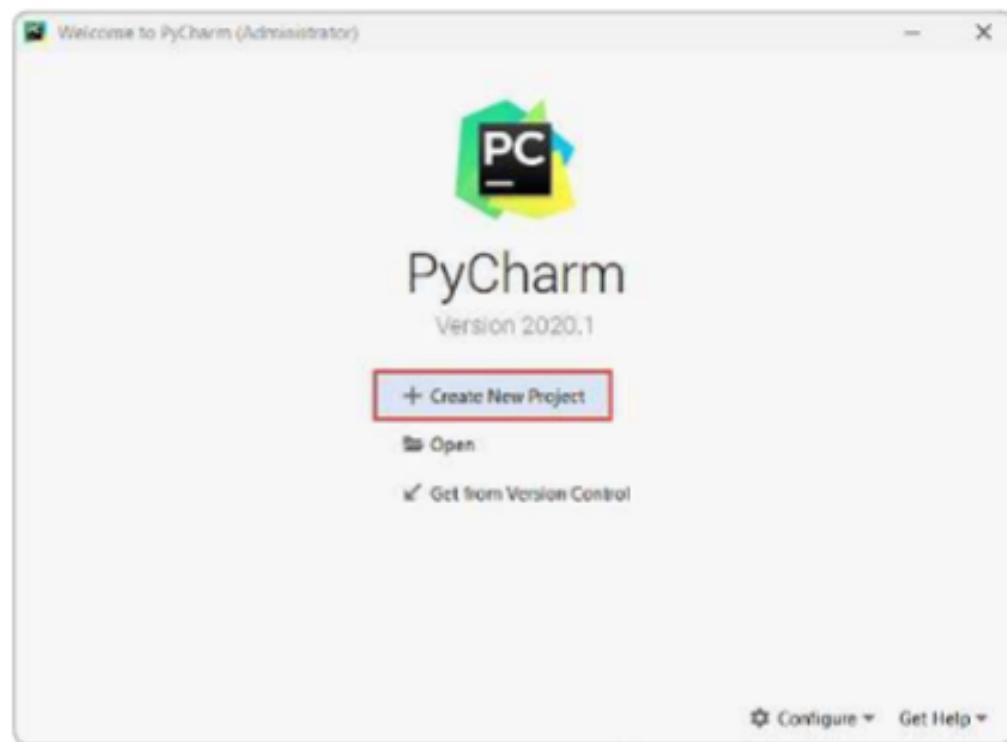
La letra "n" colocada después de la barra invertida proviene de la palabra newline (nueva línea).

1.10 Crear el proyecto Formato

Ahora, se procede a crear el proyecto Formato.

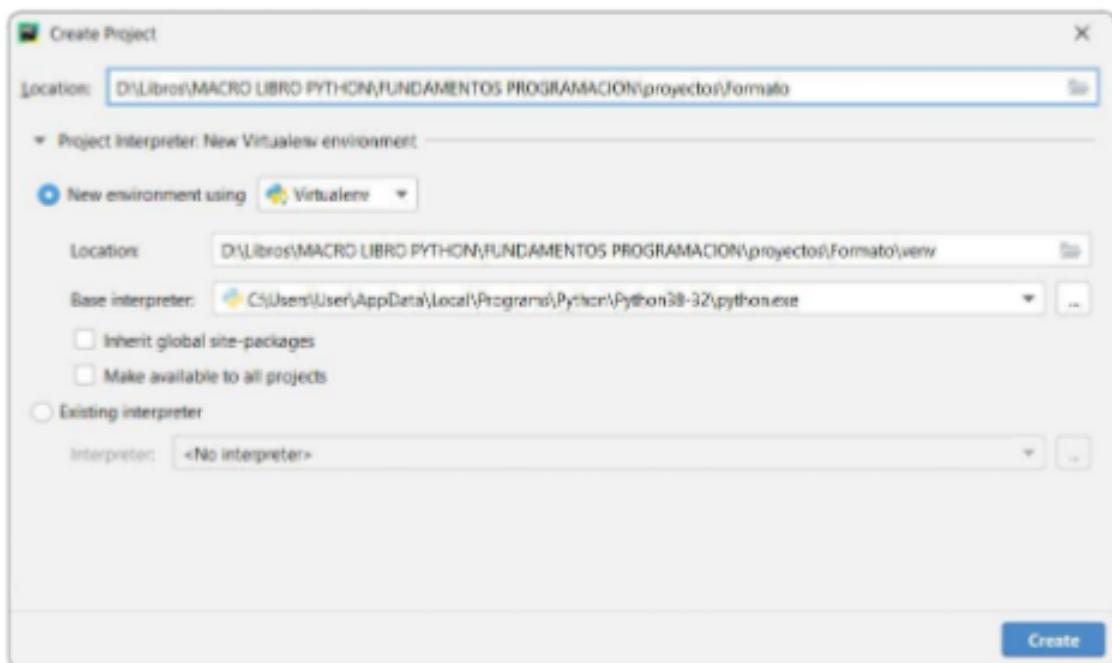
Paso 1:

Haga clic en **+ Create New Project**.



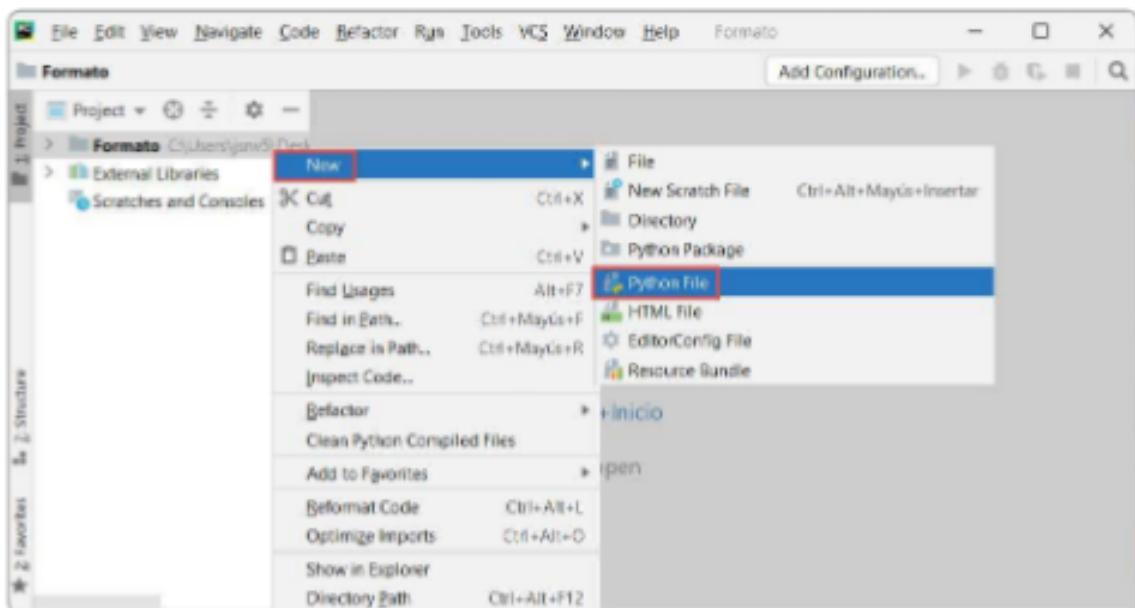
Paso 2:

Escriba el nombre del proyecto: **Formato**. Luego, haga clic en **Create**.



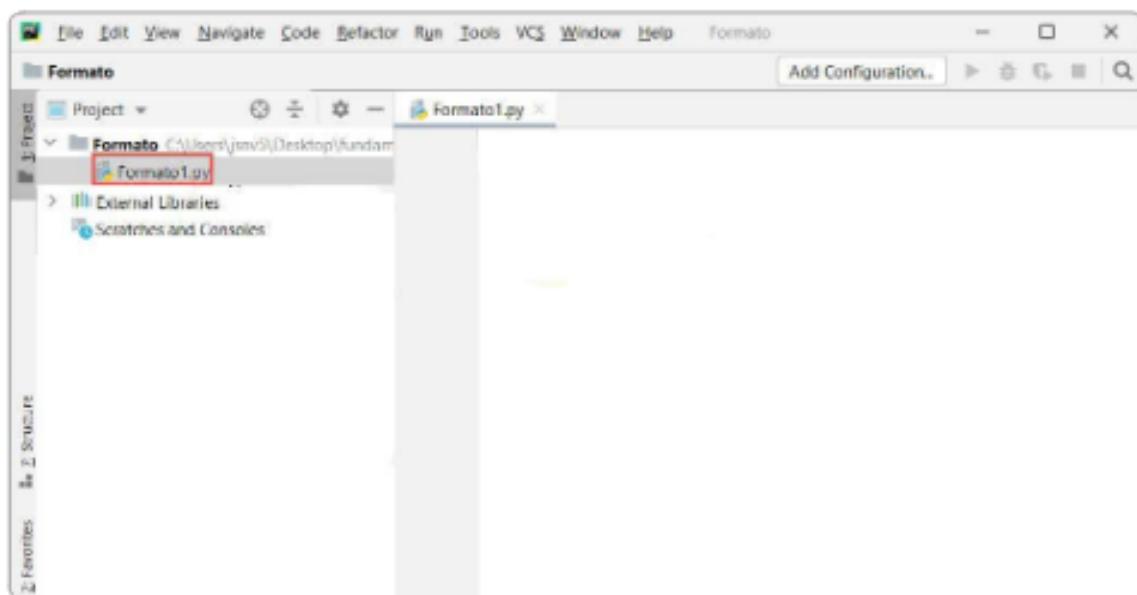
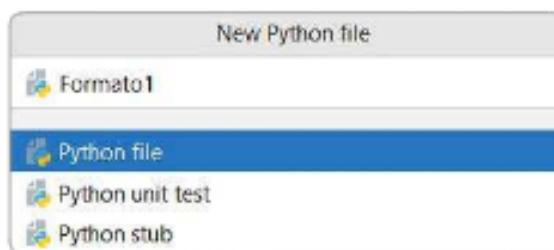
Paso 3:

Ahora, debe crear un archivo nuevo. Para ello, haga clic derecho en la carpeta creada, luego en **New** y en **Python File**. Posteriormente, nombre el archivo.



Paso 4:

Escriba el nombre del archivo: **Formato1**. Luego, presione <Enter>.



Paso 5:

Escriba el código.

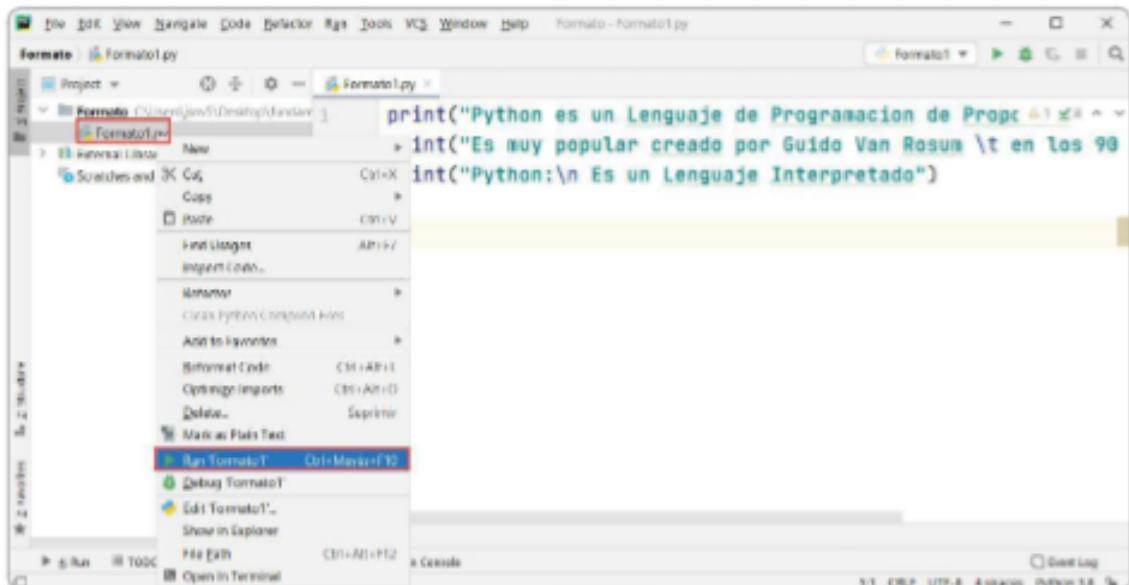
```

print("Python es un Lenguaje de Programación de Propósito General")
print("Es muy popular creado por Guido Van Rossum \t en los 90 Trabajo en Google\n")
print("Python:\n Es un Lenguaje Interpretado")

```

Paso 6:

Para ejecutarlo, haga clic derecho sobre el archivo **Formato1.py** y seleccione **Run "Formato1"**.



The screenshot shows the PyCharm interface with the terminal window open. The output of the script is displayed:

```

C:\Users\jsnv5\AppData\Local\Programs\Python\Python38-32\python.exe "C:/Users/jsnv5/Desktop/Formato1.py"
Python es un Lenguaje de Programacion de Proposito General
Es muy popular creado por Guido Van Rossum      en los 90 Trabajo en Google

Python:
    Es un Lenguaje Interpretado

Process finished with exit code 0
    
```

Nota

En el código anterior se utilizó la siguiente secuencia de escape:

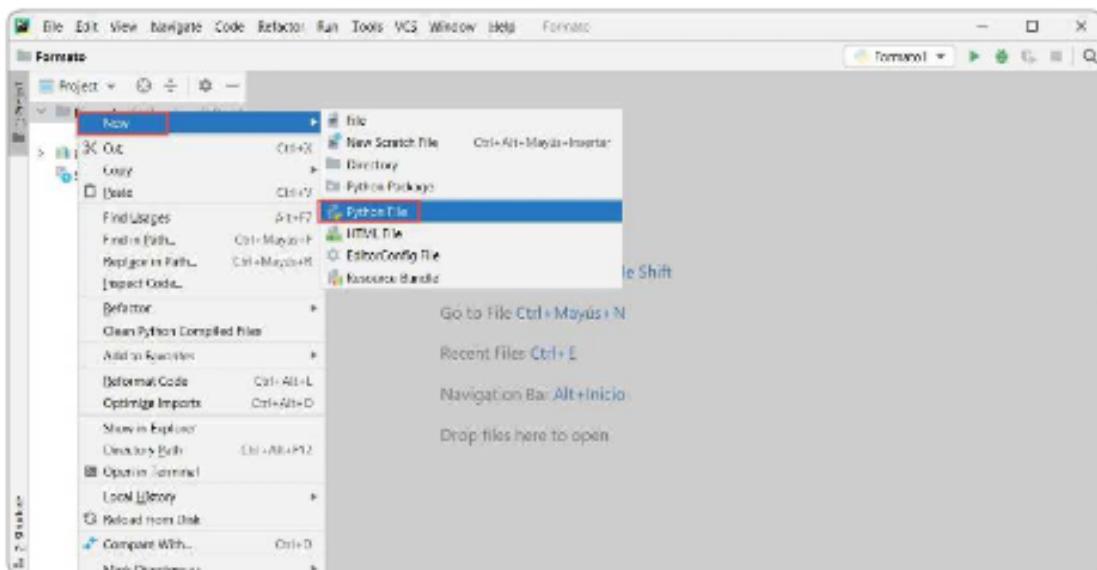


\t = Tabulación horizontal

\n = Salto de línea

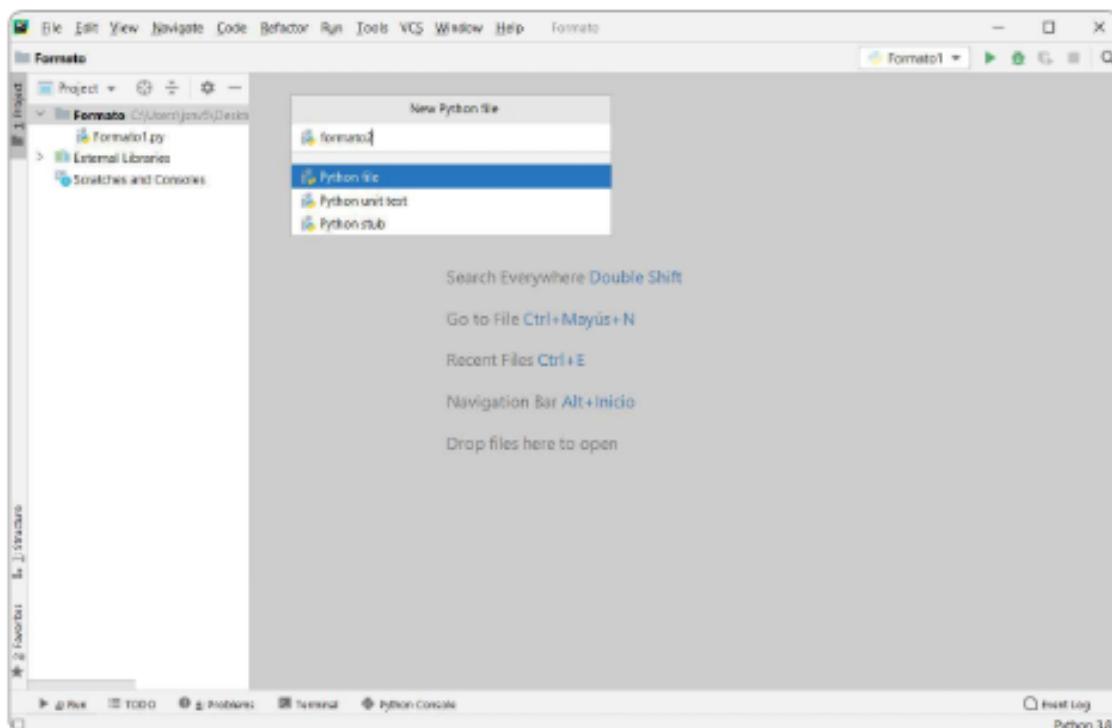
Paso 7:

Ahora debe crear un archivo nuevo. Para ello, haga clic derecho en la carpeta creada, luego en **New** y en **Python File**. Posteriormente, nombre el archivo.



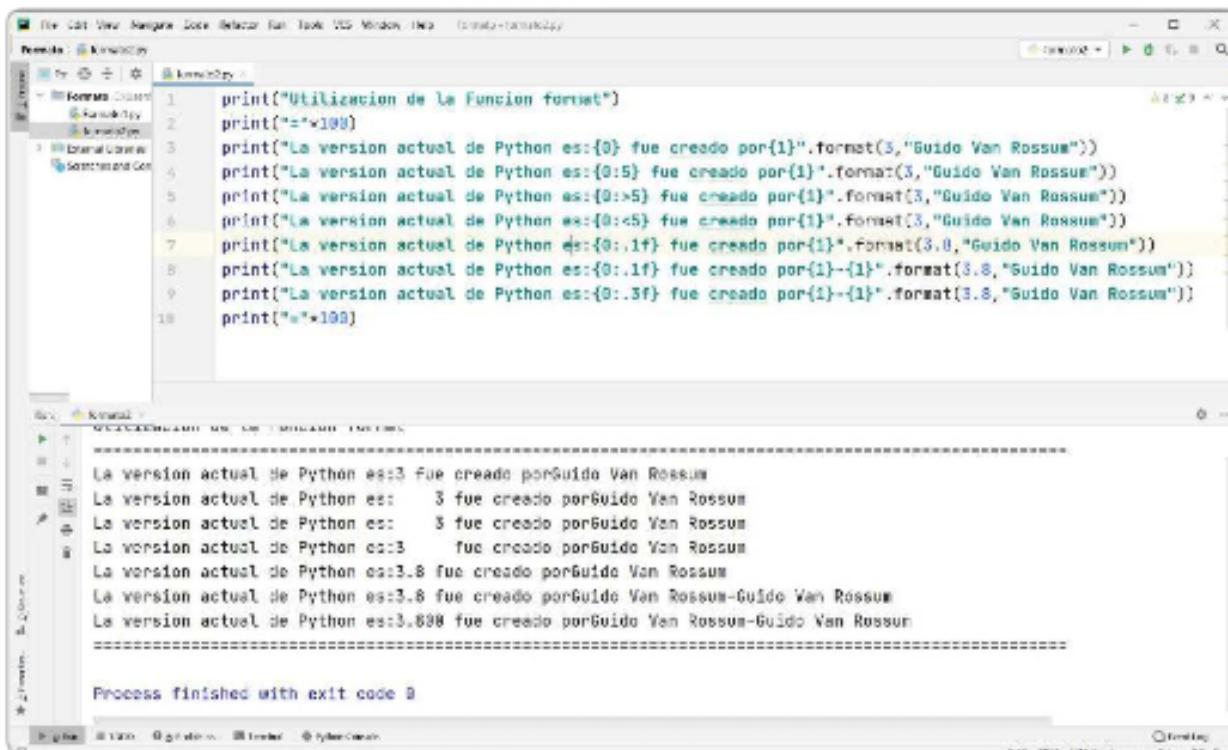
Paso 8:

Escriba el nombre del archivo: **Formato2**. Luego, presione <Enter>.



Paso 9:

Escriba el código y ejecute.



The screenshot shows the PyCharm IDE interface. In the top navigation bar, the tabs 'File', 'Edit', 'View', 'Navigate', 'Code', 'Refactor', 'Run', 'Tools', 'VCS', 'Version', and 'Help' are visible. Below the tabs, there's a search bar and a 'Format' dropdown menu. The main window displays a code editor with a file named 'Kiwisolver.py'. The code contains several print statements demonstrating the use of the `format` function with various specifiers. Below the code editor is a terminal window showing the execution results. The terminal output includes multiple lines of text where each line starts with 'La version actual de Python es:' followed by a different format specifier (e.g., `{0}`, `{0:5}`, `{0:>5}`, `{0:<5}`, `{0:.1f}`, `{0:3.1f}`, `{0:3.3f}`) and the resulting string 'fue creado por Guido Van Rossum'. At the bottom of the terminal window, the message 'Process finished with exit code 0' is displayed.

```

print("Utilización de la Función format")
print("=="*100)
print("La version actual de Python es:{0} fue creado por{1}".format(3,"Guido Van Rossum"))
print("La version actual de Python es:{0:5} fue creado por{1}".format(3,"Guido Van Rossum"))
print("La version actual de Python es:{0:>5} fue creado por{1}".format(3,"Guido Van Rossum"))
print("La version actual de Python es:{0:<5} fue creado por{1}".format(3,"Guido Van Rossum"))
print("La version actual de Python es:{0:.1f} fue creado por{1}".format(3.0,"Guido Van Rossum"))
print("La version actual de Python es:{0:3.1f} fue creado por{1}-{1}".format(3.8,"Guido Van Rossum"))
print("La version actual de Python es:{0:3.3f} fue creado por{1}-{1}".format(3.8,"Guido Van Rossum"))
print("=="*100)

Process finished with exit code 0

```

1.11 Variables

Sintaxis:

NombreVariable=Valor

Cuando se escribe algunos programas, se tiene la necesidad de almacenar valores. Para poder utilizarlos en el momento adecuado, es necesario definir y manipular las variables.

Por ejemplo, si se quiere vender *smartphone* y tabletas, en el programa se debe definir las siguientes variables:

- Número de tabletas
- Número de *smartphone*
- Precio de venta de tabletas
- Precio de venta de *smartphone*
- Monto de la venta de tabletas
- Monto de la venta de *smartphone*

Una variable es una ubicación de almacenamiento en la computadora. Cada variable tiene un nombre y un valor.



Nota



Los nombres de las variables pueden empezar por una letra o un guion bajo seguido de más letras, números o guiones bajos.

1.12 Constantes

Una constante es aquel cuyo valor no debe ser cambiado después de que se le haya asignado un valor inicial. Es una práctica común especificar una constante con el uso de letras mayúsculas.

MI_CONSTANTE = 10

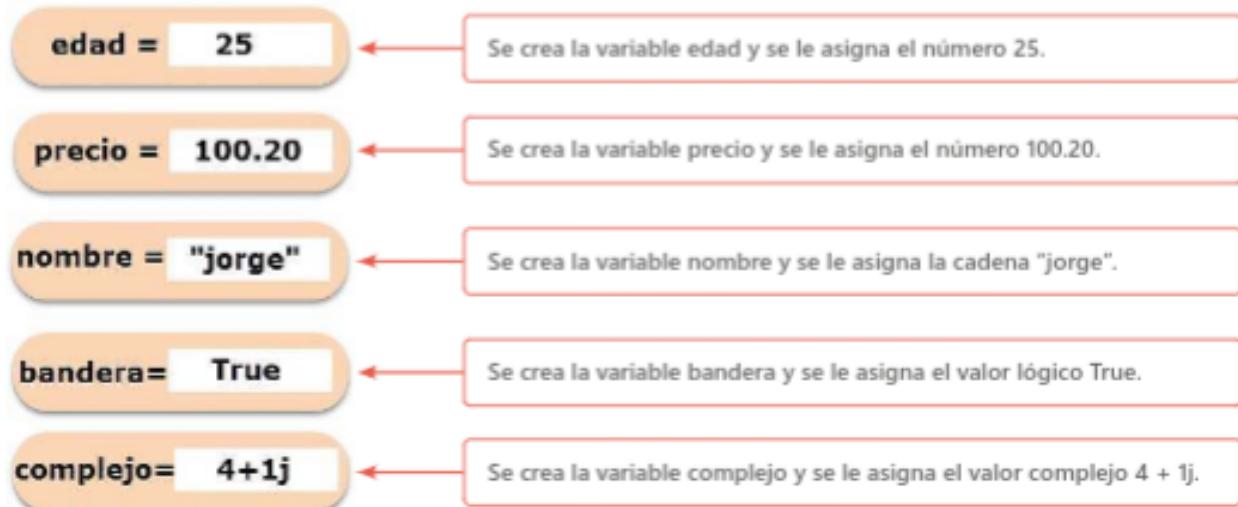
1.13 Tipos básicos

En Python, los tipos básicos se dividen en los siguientes:

- Números, como pueden ser 25 (entero), 100.20 (de coma flotante) o 4 + 1j (complejos).
- Cadenas de texto, como "jorge".
- Valores booleanos: true (verdadero) y false (falso).

Python es de tipado dinámico, esto indica que no es necesario declarar el tipo de dato que va contener la variable.

A continuación, se crearán algunas variables donde se demuestre el uso de los tipos básicos.



Tipo dato	Descripción
Numéricos	Número enteros Es expresado de la siguiente manera: Decimal: 30.60 Binario: 0b1100100 Hexadecimal: 0x3cf4 Octal: 064
	Números de punto flotante Es expresado de la siguiente manera: 0.4 Pero no olvide que puede omitir cero cuando es el único dígito delante o después del punto decimal: .4 3.1415 4.0 Por ejemplo, el valor de 4.0 se puede escribir: 4.
	Números complejos Es expresado de la siguiente manera: 6.32 + 45j 0.117j (2 + 0j) 1j

	<p>Números científicos</p> <p>Para evitar escribir tantos ceros del siguiente número:</p> <p>700000000</p> <p>La forma abreviada sería:</p> <p style="text-align: center;">8 7 x 10</p> <p>En Python, el mismo efecto se logra de una manera ligeramente diferente. Observe:</p> <p>7E8</p> <p>La letra E (también puede usar la letra minúscula e) proviene de la palabra exponente.</p>												
Cadenas	<p>Valores lógicos</p> <p>El tipo booleano es una especie de tipo numérico que es utilizado para evaluar expresiones lógicas.</p> <p>En una expresión lógica, si el resultado es verdad (true), lo contrario es false.</p> <p>Nota</p>  <p>False es numéricamente 0 True es numéricamente 1</p> <p>Las cadenas de caracteres son secuencias de caracteres entre comillas (" ") o apóstrofes (' ') indistintamente. Ejemplo: 'Hola a todos' "Vamos"</p> <p>None El tipo None representa un valor "vacío".</p> <p>Secuencia de escape Las secuencias de escape permiten introducir caracteres especiales, escapándolos (forzándolos a ser caracteres sin significado especial) con una contra barra (\) delante. A continuación, se muestra una lista de secuencias de escape:</p> <table border="1" data-bbox="668 1670 1134 1932"> <thead> <tr> <th>Secuencia</th> <th>Significado</th> </tr> </thead> <tbody> <tr> <td>\\</td> <td>Backslash</td> </tr> <tr> <td>\'</td> <td>Comilla simple</td> </tr> <tr> <td>\"</td> <td>Comilla doble</td> </tr> <tr> <td>\n</td> <td>Nueva línea</td> </tr> <tr> <td>\r</td> <td>Retorno de carro</td> </tr> </tbody> </table>	Secuencia	Significado	\\	Backslash	\'	Comilla simple	\"	Comilla doble	\n	Nueva línea	\r	Retorno de carro
Secuencia	Significado												
\\	Backslash												
\'	Comilla simple												
\"	Comilla doble												
\n	Nueva línea												
\r	Retorno de carro												

1.14 Múltiple asignación

En Python, se puede asignar un mismo valor a un conjunto de variables al mismo tiempo.

```
x = y = z = 1
```

O múltiples valores a un conjunto de variables.

```
x, y, z = 1, 2, 3
```

Números es una lista con cinco elementos y se les asigna a cinco variables.

```
numeros= [1,2,3,4,5]
num1,num2,num3,num4,num5=numeros
```

1.15 Función type

La función type devuelve el tipo de dato de una variable.

A continuación, se muestran algunos ejemplos:

Tipo1.py

```
#programa : Tipo1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripcion : este programa muestra
el uso de la funcion type
"""

edad=25 #crear la variable edad y se le asigna el numero de 25
print(type(edad)) #imprime el tipo de dato de la variable edad
precio=100.20 #crear la variable precio y se le asigna el numero de 100.20
print(type(precio)) #imprime el tipo de dato de la variable precio
nombre="jorge" #crear la variable edad y se le asigna la cadena jorge
print(type(nombre)) #imprime el tipo de dato de la variable nombre
bandera=True #crear la variable bandera y se le asigna el valor logico True
print(type(bandera)) #imprime el tipo de dato de la variable bandera
complejo=4+1j #crear la variable complejo y se le asigna el valor complejo de 4+1j
print(type(complejo)) #imprime el tipo de dato de la variable complejo

<class 'int'>
<class 'float'>
<class 'str'>
<class 'bool'>
<class 'complex'>
```

Tipo2.py

```
#programa : Tipo2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripcion : este programa muestra
el uso de la funcion type
"""

#la funcion print realizara la conversion automatica a decimal
print(type(0o123)) #octal
print(type(0x123)) #hexadecimal
print(0b110) #binario
print(3E8) # 3x10 elevado a 8
print(6.62607E-34) # 6.62607 x 10 elevado a -34.
print(0.00000000000000000000000001) # 1e-22
```

```
<class 'int'>
<class 'int'>
6
300000000.0
6.62607e-34
1e-22
```

1.16 Conversión de datos

Python ofrece dos simples funciones para especificar un tipo de dato y resolver este problema: int() y float().

La función int() toma un argumento (por ejemplo, una cadena: int(string)) e intenta convertirlo a un valor entero; si llegase a fallar, el programa entero también fallará (existe una manera de solucionarlo que se explicará más adelante).

La función float() toma un argumento (por ejemplo, una cadena: float(string)) e intenta convertirlo a flotante (el resto es lo mismo).

Ejemplo1.py

```
#programa : Ejemplo1.py
#autor : jorge nolasco valenzuela
#fecha : 1-5-2020
"""

descripcion : este programa muestra
las conversiones de datos
"""

numero = float(input("Inserta un número: "))
resultado = numero ** 3.0
print(numero, "al cubo es", resultado)
```

```
Inserta un número: 2
2.0 al cubo es 8.0
```

Ejemplo2.py

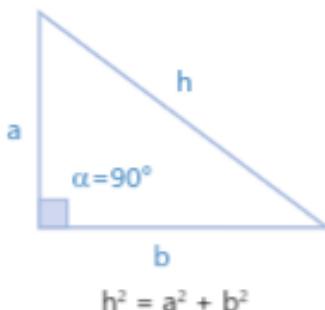
```
#programa : Ejemplo2.py
#autor : jorge nolasco valenzuela
#fecha : 1-5-2020
"""
descripcion : este programa muestra
las conversiones de datos
soles a dolares
"""

soles = float(input("Inserta soles a convertir: "))
tipo = float(input("Inserta tipo de cambio: "))
resultado = soles/tipo
print(soles, " a dolares :", resultado)
```

```
Inserta soles a convertir: 100
Inserta tipo de cambio: 3.38
100.0 a dólares : 29.585798816568047
```

1.17 Resolución de problemas matemáticos

A continuación, se construirán algunos programas para resolver algunos problemas matemáticos. Por ejemplo, en el teorema de Pitágoras, "el cuadrado de la hipotenusa es igual a la suma de los cuadrados de los otros dos lados".



a=25

b=50

Entonces, la solución será la siguiente:

$$C=(a ** 2 + b ** 2) ** 0.5$$

La salida:

```
print("c=",c)
```

pitagoras.py

```
#programa : pitagoras.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de python en resolucion de
problemas matematicos
"""
a=25 #crear la variable a y se le asigna el numero 25
b=50 #crear la variable b y se le asigna el numero 50
c=(a ** 2 + b ** 2) ** 0.5
print("c=",c)
c = 55.90169943749474
```

Nota

Puede hacer uso del `**`operador para evaluar la raíz cuadrada.

1.18 Otros tipos

Las listas son uno de los tipos de colección que posee el lenguaje Python. Sus características son las siguientes:

- Son dinámicas, es decir, pueden aumentar y disminuir su tamaño.
- Las listas se crean con dos corchetes y, dentro de ellos, se colocan a los elementos separados por comas (,). Estos pueden ser de cualquier tipo, incluso listas.

Ejemplo:

```
lista = [13, "abc", 20.35, 5+7j, [False, "ana",12]]
```

- Pueden concatenarse con otras listas usando el operador `+` y multiplicar su tamaño:

```
l= [1, 2, 3,4]
```

```
g= [5,7,2]
```

```
h= l + g    # [1,2,3,4,5,7,2]
```

```
k= g*2      # [5, 7, 2, 5, 7, 2]
```

- Son objetos mutables, es decir, es posible acceder a sus elementos y modificarlos o borrarlos con `del`. Cada índice de la lista hace referencia a un elemento, empezando por el índice 0 que referencia al primero.

```
i=[“b”,4,10,”c”]
```

```
print (i[1])    #muestra 4
```

```
pantalla[0]=1# [1, 4, 10, “c”]
```

```
del i[2]      # [1, 4, “c”] se borró el tercer elemento.
```

1.19 Comentarios

Los programas deben ser comentados para tener un mejor entendimiento de los mismos. En Python, se puede realizar comentarios de dos maneras:

```
#esto es un comentario de una linea
"""
esto es un comentario
para mas de una linea
"""


```

A continuación, se muestran algunos ejemplos que se utilizará para comentar los programas:

```
#programa : nombre.py
#autor : nombre del autor
#fecha : 01-05-2020
edad=20 # edad de una persona
"""

descripcion : este programa
muestra el uso de.....
"""


```

Nota



Los comentarios en la misma línea deben separarse con dos espacios en blanco. Luego del símbolo # debe ir un solo espacio en blanco.

1.20 Operaciones con cadena

1.20.1 Concatenación de cadenas



Cuando se tienen dos cadenas, se puede concatenar a una el resultado. Esto consiste en todos los caracteres de la primera cadena, seguido por todos los caracteres en la segunda cadena. En Python, se utiliza el operador + para concatenar dos cadenas.

```
cad = cad1 + cad2
```

cadena1.py

```
#programa : cadena1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripción : este programa muestra
el uso de cadenas
"""

nombre="Juan" #crear la variable nombre y se le asigna la cadena Juan
apellidos="Perez Perez" #crear la variable apellidos y se le asigna la cadena Perez Perez
#concatenación de cadenas
cadena3=nombre+" "+apellidos
#mostrar el nombre y apellidos
print(cadena3)
Juan Perez Perez
```

1.20.2 Multiplicar una cadena por un número

Al multiplicar una cadena por un número k, el resultado es la concatenación del mismo por las k veces.

```
cad = cad1 + cad2
```

cadena2.py

```
#programa : cadena2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripción : este programa muestra
el uso de multiplicar una cadena
por un numero
"""

cadm1="*20
cadm2="*30
cadm3="*40
#mostrar cadm1,cadm2,cadm3
print(cadm1)
print(cadm2)
print(cadm3)

=====
=====
```

1.20.3 Longitud de una cadena

Al contar el número de caracteres de una cadena, se incluye los espacios en blanco.

```
len(cadena)
```

cadena3.py

```
#programa : cadena3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de otras operaciones
de cadena
"""

nombre="pedro"
apellido="lara avila"
direccion="Av xyz 233"
#mostrar la longitud de las cadenas
print(len(nombre))
print(len(apellido))
print(len(direccion))
print("=*5)
#mostrar el nombre en forma vertical
for caracter in nombre:
    print(caracter)
print("=*5)
"""

acceso de posicion de
elementos de una cadena
"""

print(nombre[0])
print(nombre[1])
print(nombre[2])
print(nombre[3])
print(nombre[4])
print("=*5)
```

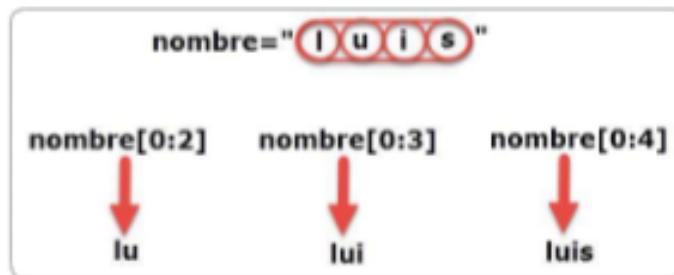
```
5
10
10
=====
p
e
d
r
o
=====
p
e
d
r
o
=====
```

1.20.4 Manejo de segmentos de una cadena

Un segmento de una cadena recibe el nombre de slice y se puede seleccionar con el siguiente operador:

[n:m]

Que devuelve parte de una cadena desde el n-esimo carácter hasta el m-esimo carácter.



A continuación, se presentan algunos ejemplos:

cadena4.py

```
#programa : cadena4.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
segmentos de cadena
"""

nombre="pedro"
apellido="lara avila"
direccion="Av xyz 233"
print("=*5)
#mostrar segmentos de una cadena
#from el indice 0 al 2
print(nombre[0:2])
#from el indice 0 al 2 de manera resumida
print(nombre[:2])
#from el indice 3 al 4
print(nombre[3:4])
#from el indice -10 al -8
print(apellido[-10:-8])
#from el indice -7 al -4
print(direccion[-7:-4])
print("=*5)
```

=====

pe

pe

r

la

xyz

=====

1.20.5 Operador in

Es el operador de pertenencia.

n in cadena

cadena5.py

```
#programa : cadena5.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso del operador in de pertenencia
"""

#n pertenece a la cadena naranja
print("n" in "naranja")
#a pertenece a la cadena naranja
print("a" in "naranja")
#nz pertenece a la cadena naranja
print("nz" in "naranja")
#az pertenece a la cadena naranja
print("az" in "naranja")
```

True

True

False

False

1.20.6 Convertir mayúsculas, minúsculas y otros

Para ello, se utilizarán algunos métodos:

capitalize()

Retorna una copia de la cadena con la primera letra en mayúsculas.

lower()

Retorna una copia de la cadena en minúsculas.

upper()

Retorna una copia de la cadena en mayúsculas.

swapcase()

Retorna una copia de la cadena convertidas las mayúsculas en minúsculas y viceversa.

title()

Retorna una copia de la cadena en mayúscula los primeros caracteres.

find()

Retorna una subcadena en una cadena de caracteres.

replace()

Retorna y cambia una subcadena de una cadena.

cadena6.py

```
#programa : cadena6.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de metodos de cadenas
"""

nombremin="luis perez"
#convertir la primera letra a mayuscula
nombremay1=nombremin.capitalize()
print("Primera letra en mayuscula :"
      +nombremay1)
#convertir toda la cadena a mayuscula
nombremay2=nombremin.upper()
print("Toda la cadena en mayuscula :"
      +nombremay2)
#convertir toda la cadena a minuscula
nombremin1=nombremay2.lower()
print("Toda la cadena en minuscula :"
      +nombremin1)
#Convertir mayúsculas a minúsculas y viceversa
nombre="Luis pEREZ"
nombremym=nombre.swapcase()
print("Convertir mayúsculas a minúsculas :"
      +nombremym)
#Convertir en forma de titulo
cadena="hola amigos"
titulo=cadena.title()
print("Titulo :" +titulo)
```

Primera letra en mayúscula :Luis perez
 Toda la cadena en mayúscula :LUIS PEREZ
 Toda la cadena en minúscula :luis perez
 Convertir mayúsculas a minúsculas :lUIS Perez
 Titulo :Hola Amigos

1.20.7 División en trozos

El método split() se utiliza para dividir una cadena en una lista.

cadena7.py

```
#programa : cadena7.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso del metodo split
"""


```

```

nombres = "jorge pedro jaime jose"
#divide los nombres en una lista
lista=nombres.split()
#muestra los elementos de una lista
for ele in lista:
    print(ele)
jorge
pedro
jaime
jose
  
```

1.20.8 Formatos de cadenas - str.format()

El formato con str.format() permite fijar la longitud de una cadena, aplicar formatos numéricos, establecer la alineación, tabular datos y llenar espacios con un determinado carácter.

```

cadena8.py
#programa : cadena8.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de formatos de cadena
"""

#variables numericas
numero1=8.67676767
numero2=9.00000001
numero3=1200.09001
numero4=90
numero5=100
numero6=110
"""

mostrar variables
con formato decimal
"""

print("{0:.2f}".format(numero1))
print("{0:f}".format(numero1))
print("{0:.2f} {1:.3f} {2:.4f}".format(numero1,numero2,numero3))
"""

mostrar variables
con formato entero
"""

print("{:d}".format(numero4))
print("{:d}".format(numero5))
print("{:d}".format(numero6))
"""

mostrar variables
con formato entero
y espaciados
"""
  
```

```

print("{:10d}".format(numero4))
print("{:9d}".format(numero5))
print("{:8d}".format(numero6))
#variables cadenas
cadena1="hola"
cadena2="amigos"
"""
mostrar cadenas
especificando el orden
"""

print("{1}{0}".format(cadena1,cadena2))
print("{1:8}{0}".format(cadena1,cadena2))
print("{0:8}{1}".format(cadena1,cadena2))
"""
mostrar cadenas
rellenando
"""

#derecha
print("{:_<10}".format(cadena1))
#izquierda
print("{:_>10}".format(cadena1))
#izquierda y derecha
print("{:_^10}".format(cadena1))

```

```

8.68
8.676768
8.68 9.000 1200.0900
90
100
110
    90
    100
    110
amigos hola
amigos hola
hola amigos
hola _____
    hola
    hola

```

1.20.9 Convertir números a cadena - str()

```

cadena9.py
#programa : cadena9.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso del metodo str
"""

```

```
#variables numericas
numero1=8.67676767
numero2=9.00000001
numero3=1200.09001
"""

mostrar numeros
convertidos a cadena
"""

print("numero1 :" +str(numero1))
print("numero2 :" +str(numero2))
print("numero3 :" +str(numero3))
```

```
numero1 :8.67676767
numero2 :9.00000001
numero3 :1200.09001
```

Nota


Si desea cadenas con comillas:

Hola soy "Jorge Nolasco"

Podrá generarlo de las siguientes formas:

```
print("Hola soy \"Jorge Nolasco\" ")
print('Hola soy "Jorge Nolasco" ')
```

```
#programa : cadena10.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""


```

```
descripcion : este programa muestra
el uso de cadenas
"""


```

```
print(" *")
print(" * *")
print(" *   *")
print(" *     *")
print(" ***   ***")
print(" *   *")
print(" *   *")
print("*****")
```

```

      *
    *   *
  *       *
*           *
***       ***
*           *
*       *
*****
*****
```

1.20.10 UTF-8: codificación de caracteres

UTF-8 codifica cada carácter como una secuencia de uno a cuatro bytes. Por ejemplo, una A sigue siendo 65, como en ASCII; pero un é es 195.

Python recoge la codificación de caracteres desde el sistema operativo (según su configuración regional).

Nota



Las cadenas son inmutables (sus elementos no se pueden modificar). Si se requieren modificaciones, se debe construir una cadena nueva.

1.21 La clase math

Esta clase permite realizar ciertos cálculos matemáticos

math1.py

```
#programa : math1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
la utilizacion de la clase math
"""

#importaremos la clase math
import math
"""

muestra la raiz
cuadrada de un
numero
"""

print(math.sqrt(4))
print(math.sqrt(9))

"""

muestra el factorial
de un numero
"""

print(math.factorial(2))
print(math.factorial(4))

"""

muestra el valor
absoluto de un numero
"""

print(math.fabs(-9))
print(math.fabs(-18))

"""

muestra un numero
redondeo hacia arriba
"""


```

```
print(math.ceil(12.98))
```

muestra un numero
redondeo hacia abajo

```
print(math.floor(12.98))
```

muestra un numero
Eleva un número x
a un exponente y

```
print(math.pow(5,3))
```

muestra el valor
de pi

```
print(math.pi)
```

muestra el logaritmo
de un numero

```
print(math.log(11))
```

2.0

3.0

2

24

9.0

18.0

13

12

125.0

3.141592653589793

2.3978952727983707

1.22 Generación de números aleatorios

El módulo random proporciona un generador de números aleatorios.

```
random.randrange([start,] stop [,step])
```

math2.py

```
#programa : math2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
```

*descripcion : este programa muestra
la generacion de numeros aleatorios*

#importaremos la clase random

```
import random
mostrara un numero aleatorios
entre 0 y 10
print(random.randrange(11))

mostrara un numero aleatorios
entre 0 y 11 con incremento de 4
print(random.randrange(1,12,4))

mostrara un numero aleatorio de
punto flotante entre 0.0 y 1.0
print(random.random())

mostrara un numero aleatorio de
punto flotante entre a y b
print(random.uniform(1,10))

mostrara un número entero entre a y b
ambos incluidos
print(random.randint(10, 20))
```

6
1
0.2248103615791086
5.775294470275517
13

1.23 Fechas y horas

El módulo que permite manipular fechas y horas de manera sencilla es el `datetime`. Es importante aclarar, que el manejo de fechas y horas está relacionado al manejo con zona horaria (`aware`) y sin zona horaria (`naive`). A continuación, se muestra el siguiente ejemplo para un mejor entendimiento.

```
Fecha1.py
#programa : Fecha1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020

descripcion : este programa muestra
el manejo de fecha y hora
# importar el modulo para manejo fechas
```

```
from datetime import datetime
#SIN ZONA HORARIA - naive
fecha1=datetime.utcnow()
print(fecha1)
print("=" * 30)
#CON ZONA HORARIA - aware
fecha2=datetime.now()
print(fecha2)
```

2020-05-01 16:15:50.911790

=====

2020-05-01 11:15:50.911790

1.23.1 Ejemplos

A continuación, se muestran algunos ejemplos del uso de fechas respecto a un formato específico:

Fecha2.py

```
#programa : Fecha2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#####
descripcion : este programa muestra
el manejo de fecha y hora
#####

from datetime import datetime
#MANEJO CON FECHA ESPECIFICA
#formato a utilizar
formato = "%Y/%m/%d %H:%M:%S"
#fecha cadena
fecha_cadena="2017/07/27 15:35:00"
#fecha en el formato especificado
fecha=datetime.strptime\
    (fecha_cadena,formato)
print(fecha)
print("=*40)
#MANEJO CON FECHA INGRESADA
#formato a utilizar
formato1 = "%Y/%m/%d %H:%M:%S"
#fecha cadena
fecha_cadena1=input("Ingrese Fecha "
    "en el formato AA/MM/DD"
    " HORA:MINUTO:SEGUNDO==>:")
#fecha con el formato especificado
fecha1=datetime.strptime\
    (fecha_cadena1,formato1)
#muestra la fecha
print(fecha)
```

```
2017-07-27 15:35:00
=====
```

```
Ingrese fecha en el formato AA/MM/DD HORA:MINUTO:SEGUNDO==>:2020/05/01 11:18:00
2017-07-27 15:35:00
```

Fecha3.py

```
#programa : Fecha3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
```

```
descripción : este programa muestra
el manejo de fecha y hora
```

```
from datetime import datetime, time
#MANEJO CON FECHA ESPECIFICA
```

```
formato a utilizar:
```

```
%H=horas
%M=minutos
%S=segundo
```

```
formato = "%H:%M:%S"
#fecha en el formato especificado
fecha=(datetime.now())
#hora en el formato especificado
hora=fecha.time()
#muestra la hora en el formato especificado
print(hora)
```

```
11:19:09.046295
```

Fecha4.py

```
#programa : Fecha4.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
```

```
descripción : este programa muestra
el manejo de fecha y hora
```

```
from datetime import datetime, time
```

```
#MANEJO CON FECHA ESPECIFICA
```

```
#formato a utilizar
```

```
formato = "%H:%M:%S"
#fecha en el formato especificado
fecha=(datetime.now())
#año
ano=fecha.year
#mes
mes=fecha.month
```

```
#dia
dia=fecha.day
#hora
hora=fecha.time()
#muestra el año , mes , dia y hora
print("Año : ",ano)
print("Mes : ",mes)
print("Dia : ",dia)
print("Hora : ",hora)
```

Año : 2020
Mes : 5
Dia : 1
Hora : 11:20:40.437892

Formatos aplicados a fechas y horas:

%a	Nombre local abreviado de día de semana
%A	Nombre local completo de dia de semana
%b	Nombre local abreviado de mes
%B	Nombre local completo de mes
%c	Representación local de fecha y hora
%d	Día de mes 01,31
%H	Hora (horario 24 horas)
%I	Hora (horario 12 horas)
%j	Número de día del año 001,366
%m	Mes 01,12
%M	Minuto 00,59
%p	Etiqueta AM o PM
%S	Segundo
%U	Número semana del año. Se considera al domingo como primer día de semana [00,53]
%w	Establece el primer día de semana [0(Domingo),1(Lunes)... 6]

%W	Número semana del año (se considera al lunes como el primer día de la semana) [00,53]
%x	Fecha local
%X	Hora local
%y	Año en formato corto 00,99
%Y	Año en formato largo
%Z	Nombre de zona horaria

Fecha5.py

```
#programa : Fecha5.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripción : este programa muestra
el manejo de fecha y hora - formatos
"""

from datetime import datetime, time

hoy = datetime.now()
#algunos formatos
#formato1 - Friday October 2019 18:26:57
formato1 = "%A %B %Y %H:%M:%S"
# Aplicando formato1
fecha1=hoy.strftime(formato1)
print("formato1 :",fecha1)

#formato2 - Fri Oct 19 18:28:33
formato2 = "%a %b %y %H:%M:%S"
# Aplicando formato1
fecha2=hoy.strftime(formato2)
print("formato2 :",fecha2)

#formato3 - Friday October 2019
formato3 = "%A %B %Y"
# Aplicando formato1
fecha3=hoy.strftime(formato3)
print("formato3 :",fecha3)

#formato4 - Fri Oct 19
formato4 = "%a %b %y"
# Aplicando formato1
fecha4=hoy.strftime(formato4)
print("formato4 :",fecha4)
```

```
formato1 : Friday May 2020 11:23:11
formato2 : Fri May 20 11:23:11
formato3 : Friday May 2020
formato4 : Fri May 20
```

A continuación, se muestran algunos ejemplos de operaciones con fechas:

Fecha6.py

```
#programa : Fecha6.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#*****#
descripcion : este programa muestra
el manejo de fecha y hora - operaciones
#*****#
from datetime import datetime, time

#obtener la edad a traves de la fecha de nacimiento
#fecha actual
print("Edad de una persona")
print("=====")
hoy = datetime.now()
#fecha de nacimiento
nacimiento = datetime(1972, 5, 11, 0, 0, 0)
#calcular la edad
edad=hoy.year - nacimiento.year
#print la edad
print("Tu edad:",edad)
print("=====")

#obtener la fecha mayor
#fecha1
print("Mayor de dos fechas")
print("=====")
fecha1 = datetime(2019,1, 17, 0, 0, 0)
#fecha2
fecha2 = datetime(2019,9, 17, 0, 0, 0)
#calcular mayor entre dos fechas
if str(fecha1)>str(fecha2):
    print("Mayor",fecha1)
else:
    print("Mayor", fecha2)
print("=*30")

#diferencias de fechas
print("Diferencia de Fechas")
print("=====")
#fecha3
fecha3 = datetime(2019,10, 17, 0, 0, 0)
```

```
#fecha4
fecha4 = datetime(2019, 9, 17, 0, 0, 0)
#diferencias entre fechas
diferencia=fecha3-fecha4
print("Diferencia de Fechas",diferencia)
print("=====")
```

Edad de una Persona

```
=====
```

Tu edad: 48

```
=====
```

Mayor de dos fechas

```
=====
```

Mayor 2019-09-17 00:00:00

```
=====
```

Diferencia de Fechas

```
=====
```

Diferencia de Fechas 30 days, 0:00:00

```
=====
```

1.24 Más sobre comentarios

Es importante añadir comentarios y explicaciones a sus códigos. Por ejemplo, a continuación, se muestra el uso de comentarios.

Comentarios después de la sentencia:

```
edad=25 #se crea la variable edad y se le asigna el numero 25
precio=100.20 #se crea la variable precio y se le asigna el numero 100.20
nombre="jorge" #se crea la variable nombre y se le asigna la cadena jorge
bandera=True #se crea la variable bandera y se le asigna el valor logico True
complejo=4+1j #se crea la variable complejo y se le asigna el valor complejo 4+1j
```

Comentarios antes de la sentencia:

```
#se crea la variable edad y se le asigna el numero 25
edad=25
#se crea la variable precio y se le asigna el numero 100.20
precio=100.20
#se crea la variable nombre y se le asigna la cadena jorge
nombre="jorge"
#se crea la variable bandera y se le asigna el valor logico True
bandera=True
#se crea la variable complejo y se le asigna el valor complejo 4+1j
complejo=4+1j
```

Comentario más de una línea:

```
programa : Variable.py
autor : jorge nolasco valenzuela
fecha : 01-05-2020
Descripción : suma de dos numeros
numero1=10
numero2=20
suma=numero1+numero2
```

Nota



Los destacados desarrolladores describen cada parte importante del código.

1.25 Operadores

Operadores aritméticos

Operador	Descripción	Operación
+	adición	Var1 = 3+5
-	sustracción	Var1 = 3-2
*	multiplicación	Var1 = 3*2
/	división	Var1=10/2
//	división entera	Var1=7/2 #Var1 es 3
**	potencia	Var1=2**2
%	módulo	Var1=7%2 #Var1 es 1

Operadores lógicos

Operador	Descripción	Expresión
And	Es verdadero (true), si ambos operadores son verdaderos.	a and b
Or	Es verdadero (true), si uno de los operadores es verdadero.	a or b
not	Es verdadero (true), si el operador es falso.	Not (a and b)

Operadores especiales

Operador	Descripción	Expresión
es	Es true, si los operadores son idénticos.	a is b
is not	Es true, si los operadores no son idénticos.	a is not b
in	Es true, si el valor o variable se encuentra en la secuencia.	a in c
not in	Es true, si el valor o variable no se encuentra en la secuencia.	a not in c

Operadores relacionales

Operador	Descripción	Expresión	Significado
==	igual que	a == b	a es igual a b
!=	diferente que	a != b	a es distinto de b
<	menor que	a < b	a es menor que b
>	mayor que	a > b	a es mayor que b
<=	menor igual que	a <= b	a es menor o igual que b
>=	mayor igual que	a >= b	a es mayor o igual que b

1.26 Algo más sobre operadores

Prioridad de operadores

Considere la siguiente expresión:

```
5 + 3 * 5
```

Primero, debe multiplicar 3 por 5 y, mantenga el 15 en su memoria. Luego, agréguelo a 5, obteniendo así el resultado de 20.

El fenómeno que hace que algunos operadores actúen antes que otros, se conoce como la jerarquía de prioridades. Python define con precisión las prioridades de todos los operadores y asume que los operadores de mayor prioridad (mayor) realizan sus operaciones antes que los operadores de menor prioridad.

El enlace del operador determina el orden de los cálculos realizados por algunos operadores con igual prioridad, colocados uno al lado del otro en una expresión. La mayoría de los operadores de Python tienen enlace a la izquierda, lo que significa que el cálculo de la expresión se realiza de izquierda a derecha.

A continuación, se muestra un ejemplo (enlace de izquierda):

```
print(11 % 2 % 2)
```

Se obtiene el siguiente resultado:

```
1
```

Pero si utiliza un operador exponencial (enlace de derecha):

```
print(3 ** 2 ** 3)
```

Se obtiene el siguiente resultado:

```
6561
```

1.27 Operadores de acceso directo

En el siguiente código:

```
Valor=20
Valor=Valor + 1
Podrá usar una forma abreviada de la segunda línea:
Valor+=1
```

```
Valor=2
Valor=Valor * 2
Podrá usar una forma abreviada de la segunda línea:
Valor*=2
```

```
Valor=10
Valor=Valor / 2
Podrá usar una forma abreviada de la segunda línea:
Valor/=2
```

```
Valor=2
Valor=Valor ** 3
Podrá usar una forma abreviada de la segunda línea:
Valor**=3
```

1.28 Concatenación

El signo + (más), cuando se aplica a dos cadenas, se convierte en un operador de concatenación:

```
string + string
```

A continuación, se muestra un ejemplo:

```
concatenacion.py
#programa : concatenacion.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
***
```

descripcion : este programa muestra la concatenacion de cadenas

```
"""
nombres="jorge santiago"
apellidop="nolasco"
apellidom="valenzuela"
print(nombres+" "+apellidop+" "+apellidom)
```

jorge santiago nolasco valenzuela

Nota



El uso + para concatenar cadenas le permitirá construir salidas más precisas que con un print().

1.29 Replicando

El signo * (asterisco), cuando se aplica a una cadena y un número, se convierte en un operador de replicación.

A continuación, se muestra un ejemplo:

replicacion1.py

```
#programa : replicacion1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra la replicacion
print(20 * "-")
```

replicacion2.py

```
#programa : replicacion2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra la replicacion
print("+" + 20 * "-" + "+")
```

replicacion3.py

```
#programa : replicacion3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
```

descripcion : este programa muestra la replicacion cadenas

```
print("+" + 20 * "-" + "+")
print(("|" + " " * 20 + "|\\n") * 5, end="")
```

```
+-----+
|           |
|           |
|           |
|           |
|           |
```

replicacion4.py

*#programa : replicacion4.py
 #autor : jorge nolasco valenzuela
 #fecha : 01-05-2020*

descripcion : este programa muestra la replicacion cadenas

```
print("+" + 20 * "-" + "+")
print(("|" + " " * 20 + "|\\n") * 5, end="")
print("+" + 20 * "-" + "+")
```

```
+-----+
|           |
|           |
|           |
|           |
|           |
+-----+
```

1.30 Operador ==

El operador == (igual a) compara los valores de dos operandos. Si son iguales, el resultado de la comparación es true. Si no son iguales, el resultado de la comparación es False.

Mire la comparación de igualdad a continuación: ¿cuál es el resultado de esta operación?

```
var1 = 0
print(var1 == 00)
```

salida

True

```
var2= 2
print(var2 == 00)
```

salida

False

1.31 Operador !=

El operador de desigualdad != compara los valores de dos operandos. Aquí está la diferencia: si son iguales, el resultado de la comparación es False. Si no son iguales, el resultado de la comparación es true.

```
var1 = 0  
print(var1 != 00)
```

salida

False

```
var2= 2  
print(var2 != 00)
```

salida

True

1.32 Desplazamiento de bits

Python ofrece otra operación relacionada con los bits individuales: shifting. Esto se aplica solo a los valores de números enteros.

Los operadores de cambio en Python son un par de digrafos: <<y>>, sugiriendo claramente en qué dirección actuará el cambio.

valor << bits

valor >> bits

Por ejemplo en las siguientes operaciones:

1200 queremos multiplicarlo por 4:

1200*2=2400

1200 queremos dividirlo entre 4:

1200/2=600

Ahora con desplazamiento de bits:

```
numero = 1200
```

```
izquierda = numero << 1
```

```
derecha = numero >> 1
```

Operaciones comunes	Operaciones con desplazamiento de bits
$1200 * 2 = 2400$	$1200 << 1$
$1200 / 2 = 600$	$1200 >> 1$

A continuación, el código completo:

```
numero = 1200
izquierda = numero << 1
derecha = numero >> 1
print(numero,izquierda,derecha)
```

1200 2400 600

1.33 Tabla de prioridad de operadores

Prioridad	Operador	
1	<code>! ~ (tipo) ++ -- + -</code>	unario
2	<code>**</code>	
3	<code>* / %</code>	
4	<code>+ -</code>	binario
5	<code><< >></code>	
6	<code><<=>> =</code>	
7	<code>== !=</code>	
8	<code>&</code>	
9	<code> </code>	
10	<code>&&</code>	
11	<code> </code>	
12	<code>= += -= *= /= %= &= ^= = >>= <<=</code>	

Preguntas

Capítulo 1



1. Python es:
 un lenguaje de máquina
 un lenguaje de alto nivel
 un lenguaje natural
2. Un conjunto de comandos es:
 una lista de instrucciones
 una lista de bajo nivel
3. ¿Qué es código fuentes?
 Código de máquina ejecutado por computadoras
 Un programa escrito
4. ¿Cuál es la actual versión de Python?
 Python 2
 Python 3
 Python 1
5. ¿Cómo obtuvo su nombre Python?
 Guido van Rossum lo nombró en honor a una familia de serpientes grandes
 Guido van Rossum lo nombró en honor flying circus de Monty Python
 Guido van Rossum lo nombró en honor a un poeta de catana
6. ¿Qué extensión tienen los archivos de Python?
 py
 Pi
 pyto
7. El operador //:
 realiza una operación entera
 no existe
 realiza una división normal
8. ¿Cuál es el resultado del siguiente fragmento?

```
print(2**2**3)
```
9. ¿Cuál es el resultado del siguiente fragmento?

```
print((12%-4), (2%5), (4**3**2))
```

10. ¿Cuál es el resultado del siguiente fragmento?

```
print((-2 / 5), (2 / 5), (2 // 5), (-2 // 5))
```

11. ¿Cuál es el resultado del siguiente fragmento?

```
print((2 % -5), (2 % 5), (4 ** 3 ** 2))
```

12. ¿Cuál es el resultado del siguiente fragmento?

```
print((5 * ((35 % 13) + 101) / (2 * 14)) // 2)
```

13. ¿Cuál es el resultado del siguiente fragmento?

```
a = '2'  
b = "2"  
print(a + b)
```

14. ¿Cuál es el resultado del siguiente fragmento?

```
2 == 2  
2 == 2.  
1 == 2
```

15. ¿Qué tipos de datos son los siguientes ejemplos: "Hola", "0010"?

- () numero, cadena
- () numero, numero
- () cadena, cadena

16. ¿Qué tipos de datos son los siguientes ejemplos: "2.5", 4.0, 518, False?

- () numero, cadena, cadena, booleano
- () numero, numero, numero, booleano
- () cadena, numero, numero, booleano

17. ¿Cuál será el resultado del siguiente fragmento de código?

```
var = 100  
var = 200 + 300  
print(var)
```

18. ¿Cuál será el resultado del siguiente fragmento de código?

```
a = '2'  
b = "2"  
print(a + b)
```


2

Estructuras de control

2.1 Instrucciones de control

2.1.1 If

Todos sabemos cómo hacer preguntas. Sin embargo, ¿cómo hacer uso razonable de respuestas en Python? Para ello, se debe tener una forma que permita hacer algo si se cumple una condición y no si no se cumple.

Para tomar tales decisiones, Python ofrece una instrucción especial. Debido a su naturaleza y su aplicación, se llama instrucción condicional. Hay varias variantes de la misma. Se comenzará con el más simple. La instrucción if es una de las más usadas en la programación. La evaluación es lógica, solo si en caso la evaluación de la condición da verdad y se operará la(s) sentencia(s) interna(s). A continuación, se muestra su sintaxis y algunos ejemplos:

Sintaxis		
if condition: statements	if condition: statements else: statements	if condition: statements elif: statements else: statements

```
if true_or_not:  
    do_this_if_true
```

Esta declaración consta de los siguientes elementos:

- La **if** palabra clave
- Uno o más espacios en blanco
- Una expresión (una pregunta o una respuesta) cuyo valor puede ser
 - True (cuando su valor sea distinto de cero)
 - False (cuando sea igual a cero)
- Una nueva línea
- Una instrucción con sangría o un conjunto de instrucciones (se requiere al menos una instrucción). La sangría se puede lograr de dos maneras: insertando un número particular de espacios (la recomendación es usar cuatro espacios de sangría) o usando el carácter de tabulación

A continuación, se muestra un ejemplo:

If1.py

```
#programa:if1.py
#autor: jorge nolasco valenzuela
#fecha: 01-05-2020
"""
descripcion:este programa muestra
el uso de la instrucción if
"""

nombre="jose"
if nombre=="jose":
    print("Bienvenido Jose")
```

Bienvenido Jose

Se crea la variable nombre y se le asigna la cadena "jose".

Si el resultado es verdadero, se ejecuta esta linea.

Se evalúa si el nombre es igual a "jose", si es verdadero entra al cuerpo y se ejecuta el código.

```
if1.py
1 #programa:if1.py
2 #autor: jorge nolasco valenzuela
3 #fecha: 1-5-2020
4 """
5 descripcion:este programa muestra
6 el uso de la instrucción
7 """
8 nombre="jose"
9 if nombre=="jose":
10     print("Bienvenido Jose")
```

Nota



A diferencia de otros lenguajes, en Python los cuerpos se denotan por bloques indentados por 4 espacios, en lugar de usar las comunes llaves {} o end.

A continuación, algunos ejemplos del uso de la instrucción if:

Caso 1: Cómo identificar el mayor de dos números

If2.py

```
#programa:if2.py
#autor:Jorge Nolasco Valenzuela
#fecha:01-05-2020
"""
descripcion:este programa muestra
el uso de la instrucción if
comparando dos numeros
"""
```

```

numero1=30
numero2=10
if numero2>numero1:
    print("menor:",numero1)
    print("mayor:",numero2)
elif numero2<numero1:
    print("menor:",numero2)
    print("mayor:",numero1)
else:
    print("numeros iguales")
  
```

menor: 10

mayor: 30

Caso 2: Cómo identificar el mayor de tres números

If3.py

```

#programa:If3.py
#autor:Jorge Nolasco Valenzuela
#fecha:01-05-2020
  
```

*descripcion:este programa muestra
el uso de la instrucción if
comparando tres números*

```

numero1=30
numero2=80
numero3=50
mayornumero=numero1
if numero2>mayornumero:
    mayornumero=numero2
elif numero3>mayornumero:
    mayornumero = numero3
print("Mayor Numero : ",mayornumero)
  
```

Mayor Numero : 80

Caso 3: Cálculo del descuento

If4.py

```

#programa:If4.py
#autor:Jorge Nolasco Valenzuela
#fecha:01-05-2020
  
```

*descripcion:este programa muestra
el uso de la instrucción if
cálculo del descuento(20% del monto)
cuando el cliente realiza
compras mayores a 3000*

```

monto=10000
descuento=0
if monto>3000:
    descuento=monto*0.20
print("descuento",descuento)
descuento 2000.0

```

Caso 4: Pago de impuesto

```

If5.py
#programa:If5.py
#autor:Jorge Nolasco Valenzuela
#fecha:01-05-2020
"""

descripcion:este programa muestra
el calculo del impuesto a la renta
cuendo los ingresos son mayores a 36312
"""

ingresos=float(input("Monto de Ingresos :"))
if ingresos>36312:
    print("Pagara Impuestos")
else:
    print("No Pagara Impuestos")
Monto de Ingresos :50000
Pagara Impuestos

```

Caso 5: Cálculo si un año es bisiesto

```

If6.py
#programa:If6.py
#autor:Jorge Nolasco Valenzuela
#fecha:01-05-2020
"""

descripcion:este programa muestra
Calculo si un año es bisiesto"""
ann=int(input("ingrese año:"))
if(ann % 4 == 0 and ann % 100 != 0 or ann % 400 == 0):
    print("El año "+str(ann)+" Si es bisiesto ")
else:
    print("El año "+str(ann)+" No es bisiesto ")
ingrese año:1904
El año 1904 Si es bisiesto Pagara

```

```

If7.py
#programa:If7.py
#autor:Jorge Nolasco Valenzuela
#fecha:01-05-2020
"""

```

```

descripción:este programa muestra
Calculo si un año es bisiesto"""
ann=int(input("ingrese año:"))
if ann % 4 == 0:
    if ann % 100 == 0:
        if ann % 400 == 0:
            print('El año es bisiesto')
        else:
            print('El año no es bisiesto')
    else:
        print('El año es bisiesto')
else:
    print('El año no es bisiesto')

```

ingrese año:2020
El año es bisiesto

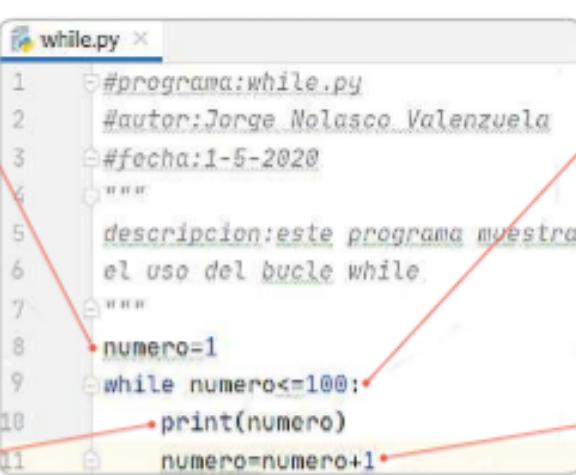
2.1.2 While

La instrucción while es un bucle controlado.

Sintaxis

while condición:
cuerpo del bucle

Por ejemplo, el siguiente programa escribe los números del 1 al 100:



```

while.py
1 #programa:while.py
2 #autor:Jorge Nolasco Valenzuela
3 #fecha:1-5-2020
4 """
5 descripción:este programa muestra
6 el uso del bucle while
7 """
8 numero=1
9 while numero<=100:
10     print(numero)
11     numero=numero+1

```

Se crea la variable numero y se le asigna el valor de 1.

Se muestra el contenido de la variable numero.

Se realiza una interacción basándose en una expresión lógica. Cuando la variable numero llegue al valor de 100, la interacción terminará.

Se incrementa en una unidad la variable numero.

A continuación, estos son algunos ejemplos del uso de la sentencia while:

while1.py

```

#programa : while1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

```

*descripcion : este programa muestra el uso de la instrucción while
Sumar todos los números del 1 al 1000*

```
total=0
contador=1
while contador<=1000:
    total=total+contador
    contador=contador+1
print("Suma:",total)
```

Suma: 500500

while2.py

*#programa : while2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020*

*descripcion : este programa muestra el uso de la instrucción while
Mostrar todos los números impares del 1 al 100*

```
contador=1
while contador<=100:
    print(contador,end=" ")
    contador=contador+2
```

**1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55 57 59 61 63 65 67
69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99**

while3.py

*#programa : while3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020*

*descripcion : este programa muestra el uso de la instrucción while
mientras no se ingrese el valor de 5*

```
valor=0
while valor!=5:
    valor=int(input("Ingrese un valor diferente a 5 :"))
    print(valor)
```

Ingrese un valor diferente a 0 :1

1

Ingrese un valor diferente a 0 :2

2

Ingrese un valor diferente a 0 :3

3

Ingrese un valor diferente a 0 :4

4

Ingrese un valor diferente a 0 :6

6

Ingrese un valor diferente a 0 :5

5

Nota

Se puede crear un ciclo infinito de la siguiente manera:

while true:

print("infinito")

2.1.3 While y else

Se puede combinar while con else. La principal ventaja es evitar el uso de variables adicionales.

A continuación, se muestra un ejemplo:

```
var1 = 1
while var1 < 10:
    print(var1)
    var1 += 1
else:
    print("else:", var1)
```

Salida

```
1
2
3
4
5
6
7
8
9
else: 10
```

2.1.4 For

Un bucle for es el que repite el bloque de instrucciones un número predeterminado de veces. A este bloque se le denomina "cuerpo del bucle" y cada repetición, "iteración".

Esta es la sintaxis de un bucle for:

```
for variable in elemento iterable (lista,cadena,range,etc):
    cuerpo del bucle
```

Por ejemplo, el siguiente programa escribe los números del 1 al 100:

```

1 #programa : for.py
2 #autor : jorge nolasco valenzuela
3 #fecha : 1-5-2020
4 """
5 descripción : este programa muestra
6 el uso del bucle for
7 """
8 for numero in range(1,101):
9     print(numero,end="")

```

1234567891011121314151617181920212223242526272829303
 1323334353637383940414243444546474849505152535455565
 7585960616263646566676869707172737475767778798081828
 384858687888990919293949596979899100

A continuación, estos son algunos ejemplos del uso del bucle for:

for2.py

```

#programa : for2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripción : este programa muestra
el uso del bucle for
imprimiendo elementos de una lista
"""

for numero in [0,1,2,3,4,5]:
    print(numero,end="")

```

012345

for3.py

```

#programa : for3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripción : este programa muestra
el uso del bucle for
imprimiendo el numero 3 tres veces
"""

for numero in [3,3,3]:
    print(numero,end="")

```

333

for4.py

```
#programa : for4.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso del bucle for
imprimiendo diversos elementos
"""

for numero in ["jose","pedro",1,2]:
    print(numero)
```

```
jose
pedro
1
2
```

for5.py

```
#programa : for5.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso del bucle for
imprimiendo en forma vertical el nombre jose
"""

for numero in "jose":
    print(numero)
```

```
j
o
s
e
```

Nota


Se puede usar comillas simples y dobles. La función range() acepta solo enteros como argumentos y genera secuencias de enteros.

For6.py

```
potencia = 1
for numero in range(16):
    print("4 elevado a ", numero, "es", potencia)
    potencia *= 4
```

```
4 elevado a 0 es 1
4 elevado a 1 es 4
4 elevado a 2 es 16
4 elevado a 3 es 64
4 elevado a 4 es 256
4 elevado a 5 es 1024
```

```
4 elevado a 6 es 4096
4 elevado a 7 es 16384
4 elevado a 8 es 65536
4 elevado a 9 es 262144
4 elevado a 10 es 1048576
4 elevado a 11 es 4194304
4 elevado a 12 es 16777216
4 elevado a 13 es 67108864
4 elevado a 14 es 268435456
4 elevado a 15 es 1073741824
```

2.1.5 For y else

Se puede combinar for con else. La principal ventaja es evitar el uso de variables adicionales.

A continuación, se muestra un ejemplo:

```
for i in range(1, 5):
    print(i)
else:
    print("else:", i)
```

Salida

```
1
2
3
4
else: 4
```

2.2 Entrada y salida estándar

La forma como los usuarios ingresan datos se da mediante la función input. Para producir salidas que dependan de las mismas, utilizan la función print.

2.2.1 Más sobre la función print

La función print() permite mostrar texto en pantalla. A continuación, algunos ejemplos:

```
print1.py
#programa : print1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#***#
descripcion : este programa muestra
el uso de la funcion print
#***#
#muestra el texto HOL
print('HOLA')
HOLA
```

print2.py

```
#programa:print2.py
#autor:Jorge Nolasco Valenzuela
#fecha: 01-05-2020
"""
descripcion:este programa muestra
el uso de la funcion print
con mas de un argumento
"""

print("HOLA","AMIGOS")
```

HOLA AMIGOS

print3.py

```
#programa:print3.py
#autor:Jorge Nolasco Valenzuela
#fecha: 01-05-2020
"""
descripcion:este programa muestra
el uso de la funcion print
para mostrar mas de una linea
"""

print("LINEA1")
print("LINEA2")
```

LINEA1

LINEA2

print4.py

```
#programa:print4.py
#autor:Jorge Nolasco Valenzuela
#fecha: 01-05-2020
"""
descripcion:este programa muestra
el uso de la funcion print
eliminando el salto de linea
"""

print("LINEA1",end="")
print("LINEA2")
```

LINEA1LINEA2

print5.py

```
#programa:print5.py
#autor:Jorge Nolasco Valenzuela
#fecha: 01-05-2020
"""
descripcion:este programa muestra
el uso de la funcion print
para imprimir variables
"""


```

```
nombre="juan"
apellido="perez"
edad=30
print(nombre)
print(apellido)
print(edad)
```

```
juan
perez
30
```

2.2.2 Cadenas f

Las cadenas f simplifican la inserción de variables y expresiones en las cadenas. Una cadena f presenta variables y expresiones entre llaves {} que se reemplazan directamente por su valor.

print6.py

```
#programa : print6.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de la funcion print con la cadenas f
"""

nombre="Juan Perez Perez"
print(f"Me presento soy {nombre} Gracias")
```

Me presento soy Juan Perez Perez Gracias

2.2.3 Break y continue

El comando break sale del bucle inmediatamente y finaliza incondicionalmente la operación del bucle. El programa comienza a ejecutar la instrucción más cercana después del cuerpo del bucle.

Por su parte, el comando continue se comporta como si el programa hubiera llegado repentinamente al final del cuerpo. Se inicia el siguiente turno y la expresión de condición se prueba de inmediato.

A continuación, un ejemplo de la sentencia break:

```
print("Instruccion break:")
for i in range(1, 6):
    if i == 3:
        break
    print("dentro del bucle ", i)
print("fuera del bucle ")
```

Salida

```
Instruccion break:  
dentro del bucle 1  
dentro del bucle 2  
fuera del bucle
```

A continuación, se muestra un ejemplo de la sentencia continue:

```
print("Instruccion continue:")  
for i in range(1, 6):  
    if i == 3:  
        continue  
    print("dentro del bucle ", i)  
print("fuera del bucle ")
```

Salida

```
Instruccion continue:  
dentro del bucle 1  
dentro del bucle 2  
dentro del bucle 4  
dentro del bucle 5  
fuera del bucle
```

2.2.4 Input

La función input() puede leer los datos ingresados por el usuario y devolverlos al programa en ejecución que puede manipularlos, pues, permite que el código sea realmente interactivo.

Esta función permite obtener texto escrito por teclado. A continuación, se muestran algunos ejemplos:

```
input1.py  
#programa : input1.py  
#autor : jorge nolasco valenzuela  
#fecha : 01-05-2020  
***  
descripcion : este programa muestra  
el uso de la funcion input  
***  
print("¿Cual es su nombre? ")  
nombre = input()  
print("Mucho Gusto,", nombre)  
  
¿Cual es su nombre?  
juan  
Mucho Gusto, juan
```

Eliminando el salto de línea:

input2.py

```
#programa : input2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de la funcion input , eliminando el salto de linea
"""

print("¿Cual es su nombre? ",end="")
nombre = input()
print("Mucho Gusto,", nombre)
```

¿Cual es su nombre? jorge
Mucho Gusto, jorge

Ahora, se modifican los ejemplos anteriores:

input3.py

```
#programa : input3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de la funcion input

"""

nombre = input("¿Cual es su nombre? ")
print("Mucho Gusto,", nombre)
```

¿Cual es su nombre?
juan
Mucho Gusto, juan



Al utilizar la función input y luego elevar al 2 el siguiente número:

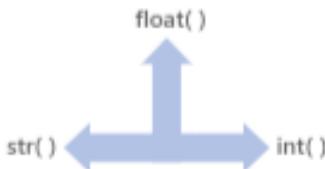
```
numero = input("Ingrese un Numero: ")
numero = numero ** 2.0
```

Obtendrá el siguiente error:

Ingrese un Numero: 3

Traceback (most recent call last):

```
  File "D:/Libros/MACRO LIBRO PYTHON/PROYECTOS/comentarios/prohibidoinput.py",
line 2, in <module>
    numero = numero ** 2.0
TypeError: unsupported operand type(s) for ** or pow(): 'str' and 'float'
```



2.2.5 Tipos de casting

Para la resolución del problema anterior, Python ofrece dos funciones simples para especificar un tipo de datos y resolver estos problemas: `int()` y `float()`.

- La función `int()` toma un argumento (por ejemplo, una cadena `int(string)`) e intenta convertirlo en un entero. Si falla, también fallará todo el programa (hay una solución para esta situación que se le mostrará más adelante).
- La función `float()` toma un argumento (por ejemplo, una cadena `float(string)`) e intenta convertirlo en flotante (el resto es el mismo).

La solución para el problema anterior será la siguiente:

```

numero = int(input("Ingrese un Numero: "))
numero = numero ** 2.0
    
```

También puede ser:

```

numero = float(input("Ingrese un Numero: "))
numero = numero ** 2.0
    
```

2.2.6 Conversión a cadena

Este tipo de conversión también puede convertir un número en una cadena, lo que es mucho más fácil y seguro. Esta operación siempre es posible.

Para convertir número a cadena, se puede utilizar la función `str()`. A continuación, se muestran algunos ejemplos:

`str(numero)`

A

```

numero1=10
numero2=20
print(str(numero1),str(numero2))
    
```

Salida

```

10 20
    
```

2.2.6.1 Entrada de números

La función `input` solo obtiene cadena de texto. Si se necesita obtener valores numéricos, se deben realizar los siguientes pasos:

```
numero = int(input ("Por favor ingrese su edad:"))
```

input3.py

```
#programa : input3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#www

descripcion : este programa muestra
el uso de la funcion input , valores numericos

numero = int(input("Por favor ingrese su edad:"))
sueldo = float(input("Por favor ingrese su sueldo:"))

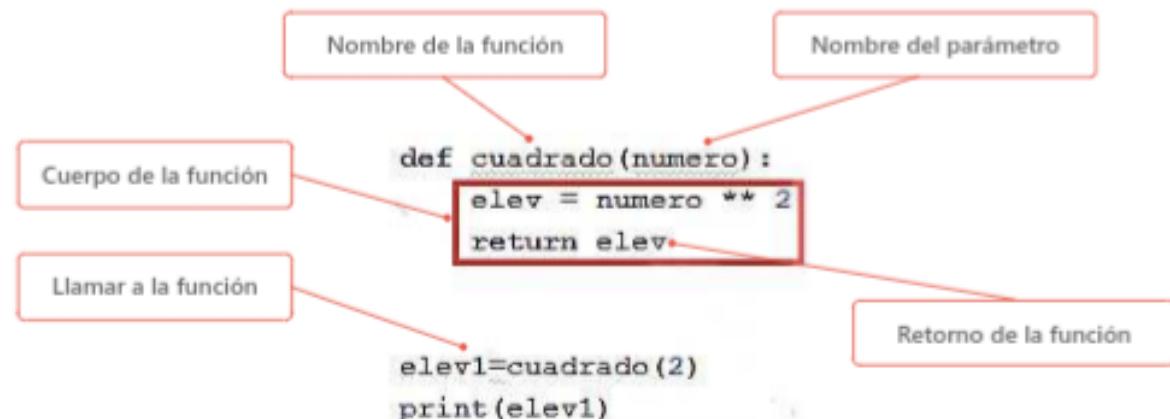
Por favor ingrese su edad:20
Por favor ingrese su sueldo:2000
```

2.3 Funciones

Las funciones son fragmentos de código asociados a un nombre, ejecutadas cuando se hacen referencia a ellas (call) y devuelven un valor. Se definen usando la palabra reservada `def`; luego el nombre que se le asigne a la función, paréntesis "()", dos puntos ":" y en la siguiente línea e indentado el cuerpo y al final la palabra reservada `return` seguido de lo que se retornara. Para ejecutar el cuerpo de la función se debe hacer referencia a su nombre.

Sintaxis

```
def function Name(parameterName1, parameterName2, . . . ) :
statements
Syntax
```



2.3.1 Crear el proyecto Funciones

Ahora, se procede a crear el proyecto Funciones

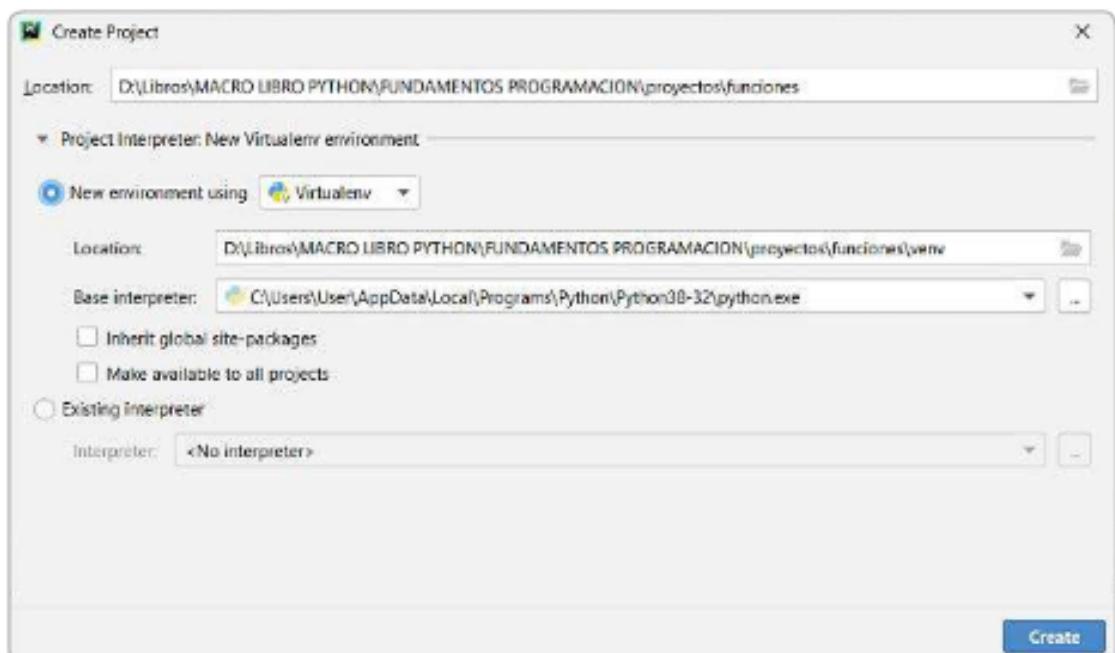
Paso 1:

Haga clic en **+ Create New Project**.



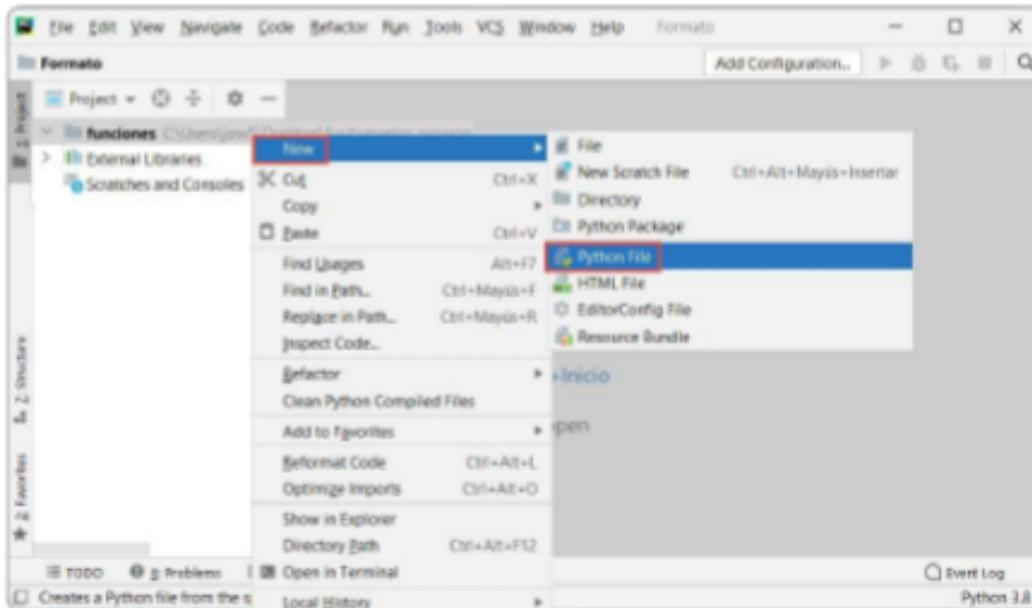
Paso 2:

Escriba el nombre del proyecto: **funciones**. Luego, haga clic en **Create**.



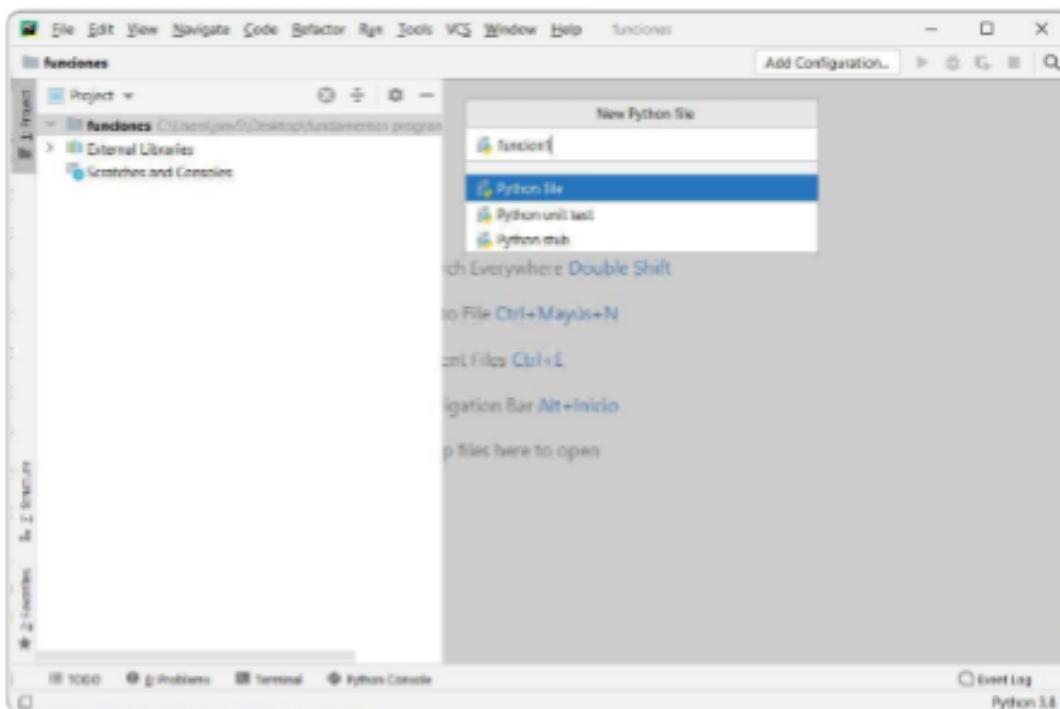
Paso 3:

Ahora, debe crear un nuevo archivo. Para ello, haga clic derecho sobre la carpeta creada **funciones**. Luego, en **New** y en **Python File**. Posteriormente, nombre el archivo.



Paso 4:

Escriba el nombre del archivo: **funcion1**. Luego, presione <Enter>.



Paso 5:

Escriba el código.

```
funcion1.py
1 #programa : funcion1.py
2 #autor : jorge nolasco valenzuela
3 #fecha : 1-5-2020
4 """
5 descripcion : este programa muestra
6 el uso de funciones
7 """
8 def cuadrado(numero):
9     elev=numero**2
10    return elev
11
```

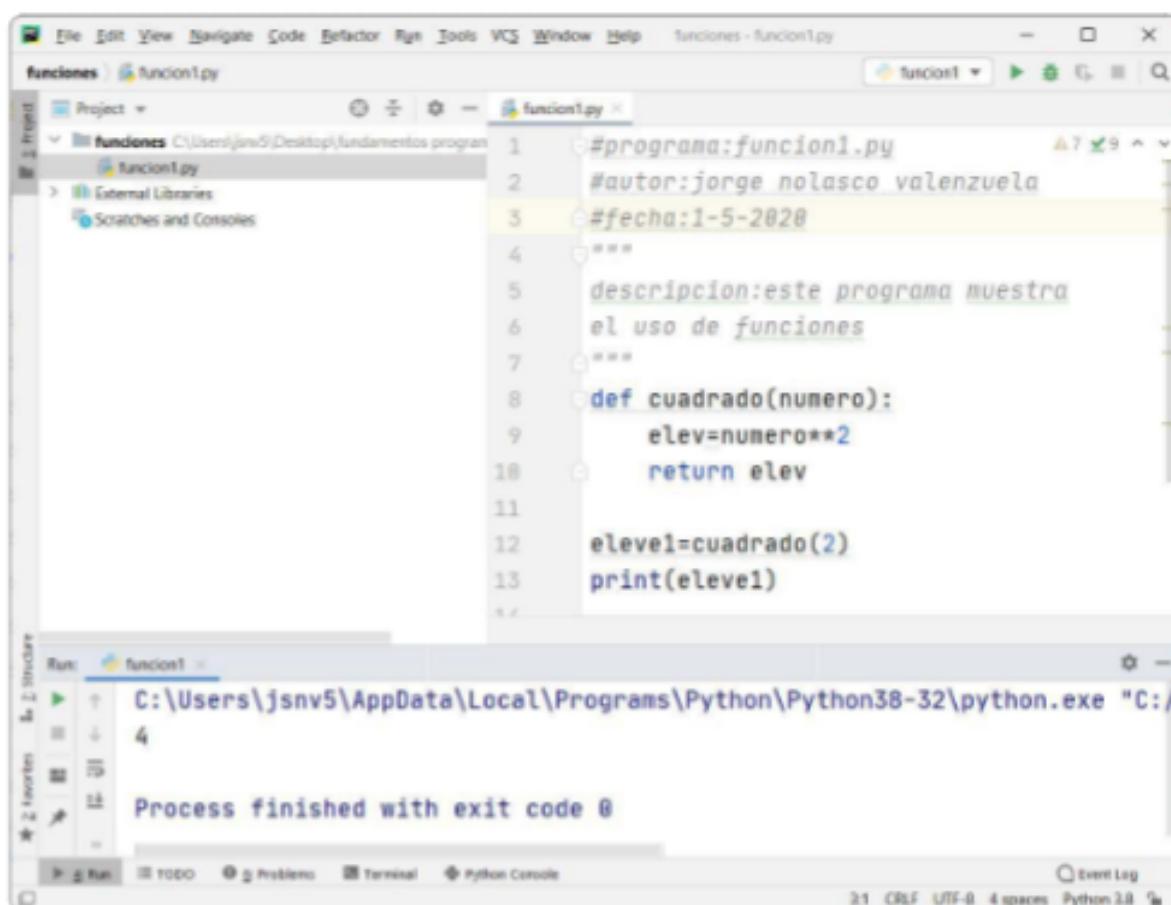
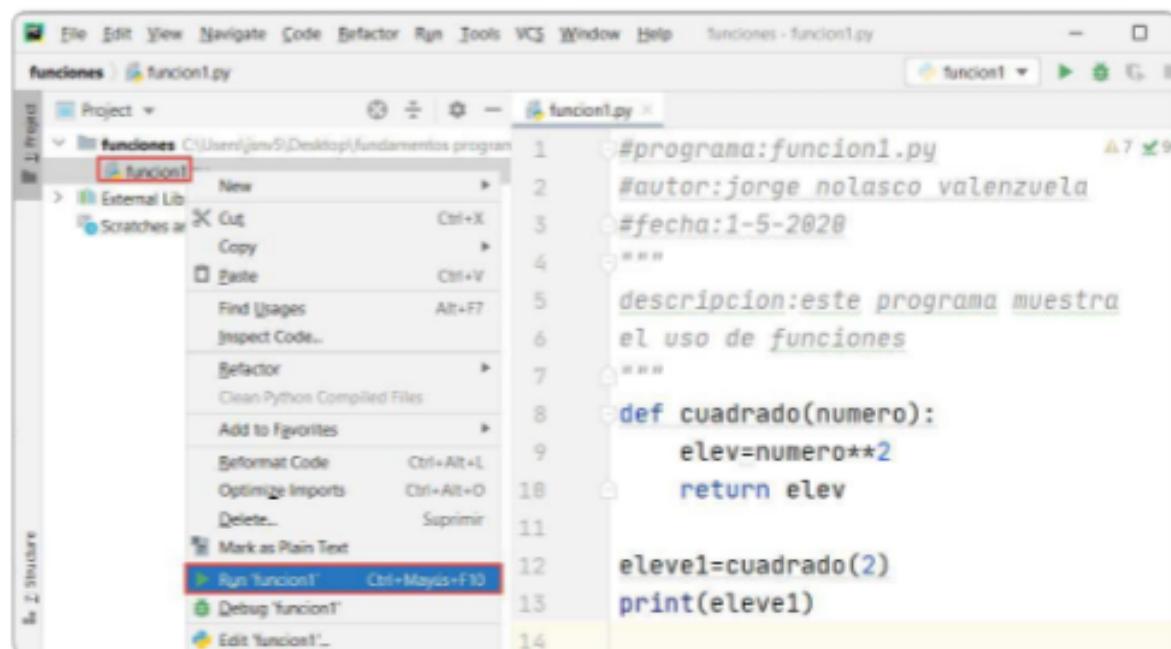
Paso 6:

Ahora, llame a la función cuadrado.

```
funcion1.py
1 #programa : funcion1.py
2 #autor : jorge nolasco valenzuela
3 #fecha : 1-5-2020
4 """
5 descripcion : este programa muestra
6 el uso de funciones
7 """
8 def cuadrado(numero):
9     elev=numero**2
10    return elev
11 eleve1=cuadrado(2)
12 print(eleve1)
```

Paso 7:

Para ejecutarlo, haga clic derecho en el archivo **funcion1.py**. Luego, seleccione **Run "funcion1"**.



Paso 8:

En Python, no es necesario definir la función principal cada vez que escribe un programa. Puede tener un punto de partida definido para la ejecución de su programa. Para ello, utilice la función **main**. Escribalo y presione <Tab>.

```

1 #programa : funcion1.py
2 #autor : jorge nolasco valenzuela
3 #fecha : 1-5-2020
4 """
5 descripcion : este programa muestra
6 el uso de funciones
7 """
8 def cuadrado(numero):
9     elev=numero**2
10    return elev
11 main

```

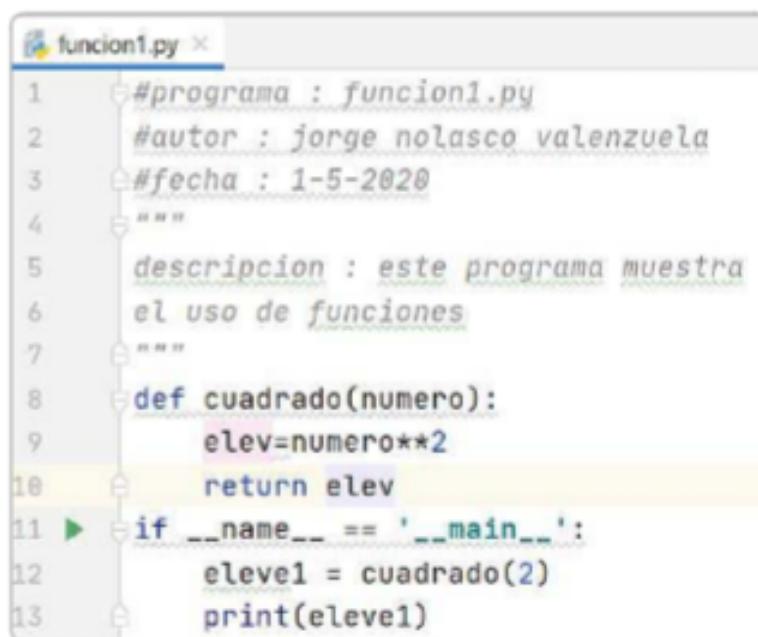
```

1 #programa : funcion1.py
2 #autor : jorge nolasco valenzuela
3 #fecha : 1-5-2020
4 """
5 descripcion : este programa muestra
6 el uso de funciones
7 """
8 def cuadrado(numero):
9     elev=numero**2
10    return elev
11 if __name__ == '__main__':

```

Paso 9:

Ahora, llame a la función cuadrado.



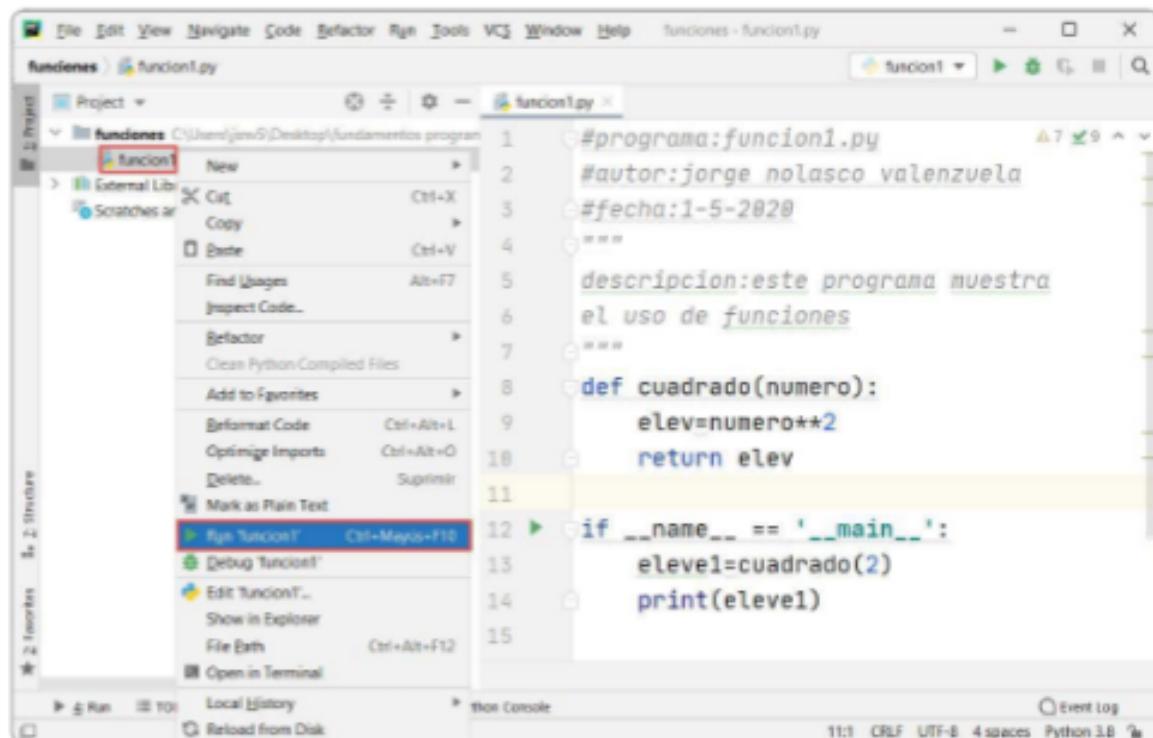
```

1 #programa : funcion1.py
2 #autor : jorge nolasco valenzuela
3 #fecha : 1-5-2020
4 """
5 descripcion : este programa muestra
6 el uso de funciones
7 """
8 def cuadrado(numero):
9     elev=numero**2
10    return elev
11 if __name__ == '__main__':
12     eleve1 = cuadrado(2)
13     print(eleve1)

```

Paso 10:

Para ejecutarlo, haga clic derecho en el archivo **funcion1.py**. Luego, seleccione **Run "funcion1"**.



```

1 #programa:funcion1.py
2 #autor:jorge nolasco valenzuela
3 #fecha:1-5-2020
4
5 descripcion:este programa muestra
6 el uso de funciones
7
8 def cuadrado(numero):
9     elev=numero**2
10    return elev
11
12 if __name__ == '__main__':
13     eleve1=cuadrado(2)
14     print(eleve1)

```

Ran: funcion1
C:\Users\jsnv5\AppData\Local\Programs\Python\Python38-32\python.exe "C:/
4
Process finished with exit code 0

2.3.2 Ejemplos

A continuación, estos son algunos ejemplos del uso de funciones:

funcion2.py

```

#programa : funcion2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de funciones , sumando 2 numeros
"""

def suma(numero1,numero2):
    return numero1+numero2
if __name__ == '__main__':
    resultado=suma(2,4)
    print(resultado)

```

6

Ejemplo de una función retornando valores y dando la bienvenida:

funcion3.py

```

#programa : funcion3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de funciones , da la bienvenida
"""

```

```
def bienvenida(nombre):
    return "Bienvenido :" + nombre
if __name__ == '__main__':
    resultado=bienvenida("Pedro")
    print(resultado)
```

Bienvenido : Pedro

Ejemplo de una función retornando valores, de elevación al cuadrado y al cubo:

funcion4.py

```
#programa : funcion4.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
'''
descripcion : este programa muestra
el uso de funciones , retornando multiples valores
elevacion al cuadrado y al cubo
'''
def elevacion(numero):
    return numero**2 , numero**3
if __name__ == '__main__':
    cuadrado , cubo = elevacion(2)
    print(cuadrado,cubo)
```

4 8

Ejemplo de una función retornando valores, de elevación al cuadrado, y de elevación al cubo ingresando el número:

funcion5.py

```
#programa : funcion5.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
'''
descripcion : este programa muestra
el uso de funciones , retornando multiples valores
elevacion al cuadrado y al cubo con valores Introducidos
'''
def elevacion(numero):
    return numero**2 , numero**3
if __name__ == '__main__':
    numero = int(input("Ingrese Numero : "))
    cuadrado , cubo = elevacion(numero)
    print(cuadrado,cubo)
```

Ingrese Numero: 2

4 8

Ejemplo de una función retornando el promedio de calificación de un alumno:

funcion6.py

```
#programa : funcion6.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020

descripción : este programa muestra
el uso de funciones , retornando el
promedio de calificación de un alumno

def promedio(ca1,ca2,ca3):
    return (ca1+ca2+ca3)/3
if __name__ == '__main__':
    calificación1 = int(input
        ("Ingrese 1 Calif.:"))
    calificación2 = int(input
        ("Ingrese 2 Calif.:"))
    calificación3 = int(input
        ("Ingrese 3 Calif.:"))
    prom = promedio(calificación1,
        calificación2,calificación3)
    print("El Promedio es :{0:5.0f}"
        .format(prom))
```

Ingrese 1 Calif.:11

Ingrese 2 Calif.:10

Ingrese 3 Calif.:9

El Promedio es : 10

Ejemplo de funciones para convertir grados Celsius a Fahrenheit:

funcion7.py

```
#programa : funcion7.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020

descripción : este programa muestra
el uso de funciones
convertir grados celsius
a fahrenheit

def convertirFahrenheit(celsius):
    fahrenheit = celsius * (9/5) + 32.0
    return fahrenheit
if __name__ == '__main__':
    celsius=int(input
        ("Ingrese Grados celsius :"))
    print("Grados fahrenheit : {0:8.3f}"
        .format(convertirFahrenheit(celsius)))
```

Ingrese Grados celsius :20

Grados fahrenheit: 68.000

Ejemplo de funciones para mostrar la tabla de multiplicar:

funcion8.py

```
#programa : funcion8.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripcion : este programa muestra
el uso de funciones
tabla de multiplicar de un
numero especifico
"""

def tablaMultiplicar(numero):
    for x in range(1,11):
        print(str(numero) + " * " + str(x)
              + " = " + str(numero * x))
if __name__ == '__main__':
    numero=int(input("Ingrese Numero :"))
    tablaMultiplicar(numero)
```

Ingrese Numero :2

```
2*1 = 2
2*2 = 4
2*3 = 6
2*4 = 8
2*5 = 10
2*6 = 12
2*7 = 14
2*8 = 16
2*9 = 18
2*10 = 20
```

Ejemplo de funciones para imprimir 20 números aleatorios entre 0 y 1:

funcion9.py

```
#programa : funcion9.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripcion : este programa muestra
el uso de funciones
generacion 20 de numeros
aleatorios entre 0 y 1
"""
```

```
import random
def aleatorios():
    for x in range(20):
        print(random.uniform(0, 1))
if __name__ == '__main__':
    aleatorios()
```

```
0.13320459491076353
0.5215462602280568
0.8774899073753551
0.11879555262887431
0.9600682930208485
0.2639084138136071
0.33075586916032285
0.7967998450151144
0.7945879776455016
0.19957832715012913
0.3639321750162231
0.30506243995746374
0.4835360033139866
0.5474816597433706
0.18526709627398485
0.7842174230467988
0.14018781790353751
0.55176292156946
0.7395893084984299
0.8315080006991621
```

2.3.3 Funciones con parámetros no definidos

funcion10.py

```
#programa : funcion10.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de funciones
con cantidad de
parametros dinamicos
"""

def datos(*args):
    print("datos pasados :",args)
if __name__ == '__main__':
    datos("jose","pedro","jaime","luis")
datos pasados : ('jose', 'pedro', 'jaime', 'luis')
```

Nota



*args lo retorna como una tupla.

funcion11.py

```
#programa : funcion11.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de funciones con cantidad de
parametros dinamicos
"""
def datos(**kwargs):
    print("datos pasados :", kwargs)
if __name__ == '__main__':
    datos(valor1="jose", valor2="pedro",
          valor3="jaime", valor4="luis")
print(datos)
print("datos pasados : ('jose', 'pedro', 'jaime', 'luis')")
```

Nota

**kwargs lo retorna como un diccionario.

2.3.4 Funciones recursivas

Una función recursiva es aquella que se llama a sí misma. A continuación, algunos ejemplos:

funcionr1.py

```
#programa : funcionr1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de funciones , utilizando recursividad
si el parametros es cero se mostrara el mensaje
Estimados Pasajeros Despegaremos
"""
def cuenta(numero):
    if numero == 0:
        print("Estimados Pasajeros Despegaremos")
    else:
        print(numero)
        cuenta(numero-1)
if __name__ == '__main__':
    cuenta(20)
```

```
20
19
18
17
16
15
14
```

```
13
12
11
10
9
8
7
6
5
4
3
2
1
```

Estimados Pasajeros Despegamos

funcionr2.py

```
#programa : funcionr2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de funciones , utilizando recursividad
sucesión de Fibonacci
"""


```

def fibonacci(numero):

```
    a, b = 0,1
    while a < numero:
        print(a,end=" ")
        a, b = b, a+b
if __name__ == '__main__':
    fibonacci(1000)
```

```
0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
```

funcionr3.py

```
#programa : funcionr3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""


```

*descripcion : este programa muestra
el uso de funciones , utilizando recursividad
limpieza de 10 habitaciones*

def limpieza(habitaciones):

if habitaciones == 0:

```
    print("Limpieza de Todas las Habitaciones")
else:
    print("Limpieza de Habitacion",habitaciones)
    limpieza(habitaciones-1)
```

```

if __name__ == '__main__':
    limpиеza(10)
Limpieza de Habitacion 10
Limpieza de Habitacion 9
Limpieza de Habitacion 8
Limpieza de Habitacion 7
Limpieza de Habitacion 6
Limpieza de Habitacion 5
Limpieza de Habitacion 4
Limpieza de Habitacion 3
Limpieza de Habitacion 2
Limpieza de Habitacion 1
Limpieza de Todas las Habitaciones

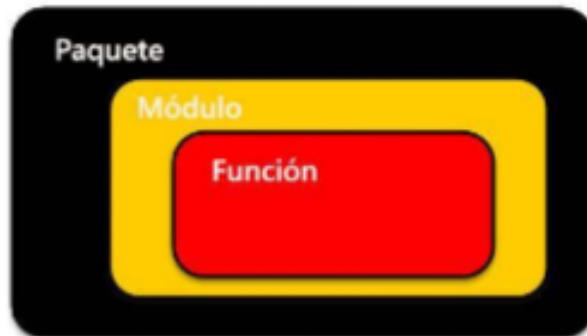
```

2.4 Módulos y paquetes

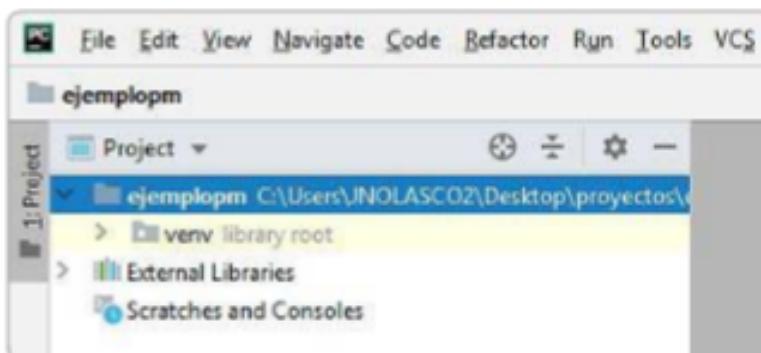
Un módulo es un tipo de contenedor que posee funciones. Puede empaquetar tantas funciones como desee en un módulo y distribuirlo.

Es buena idea no mezclar funciones con diferentes aplicaciones dentro de un módulo (al igual que en una biblioteca, nadie espera que los trabajos de matemáticas se incluyan entre los de lenguaje), así que se agrupa las funciones cuidadosamente y asigna un nombre al módulo que las represente. Es importante indicar que los módulos deberían estar agrupadas en un paquete.

Un módulo es un archivo con extensión .py que contiene funciones, clase y variables. Los módulos realizan una división lógica de los programas.

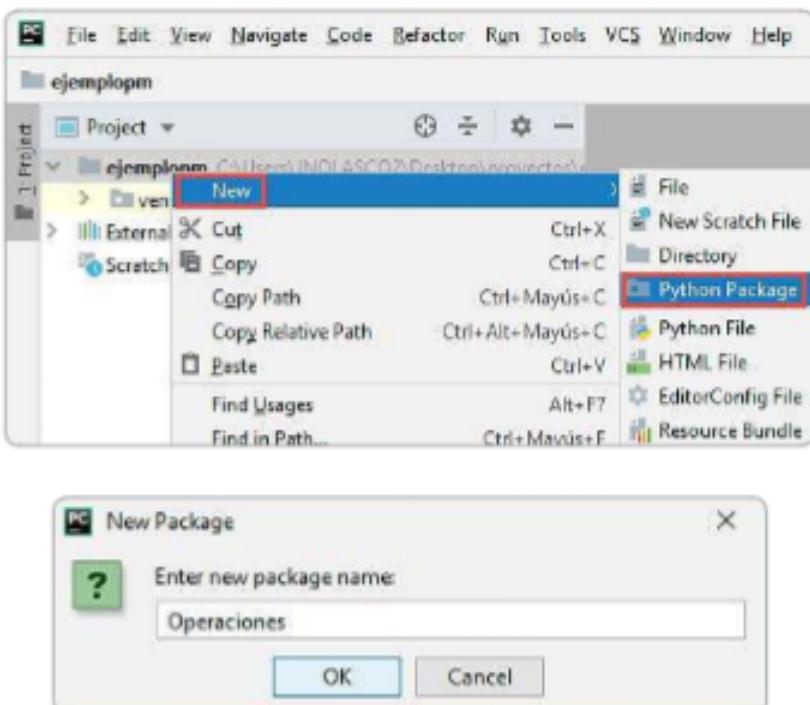


A continuación, se creará el proyecto **ejemplomp** en PyCharm:



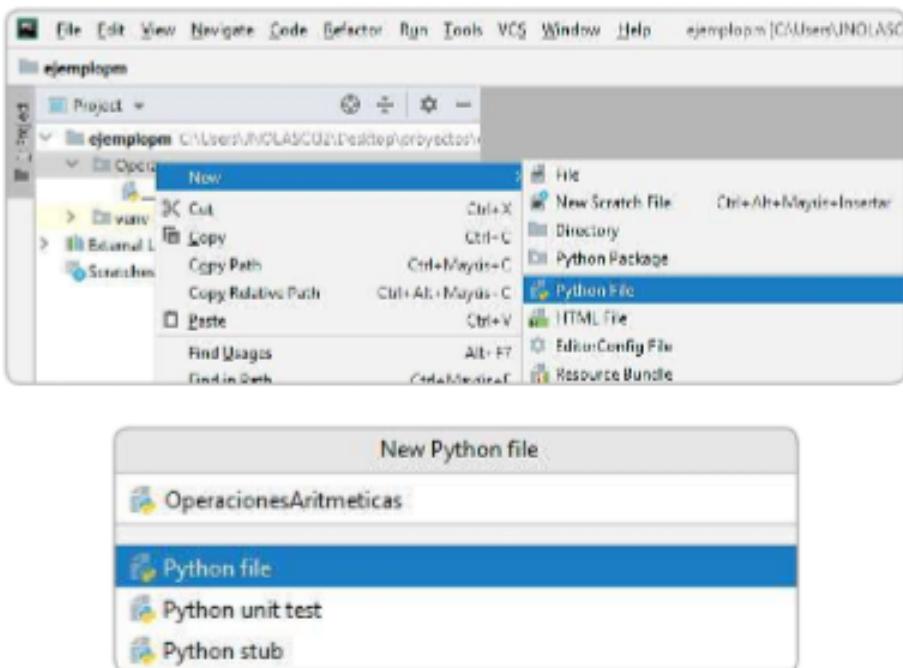
» Paquete

Ahora, se creará el paquete **Operaciones**:



» Módulos

Luego, se creará el módulo **OperacionesAritmeticas.py** dentro del paquete **Operaciones**:



OperacionesAritmeticas.py

```
#programa : OperacionesAritmeticas.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
modulo : OperacionesAritmeticas
"""

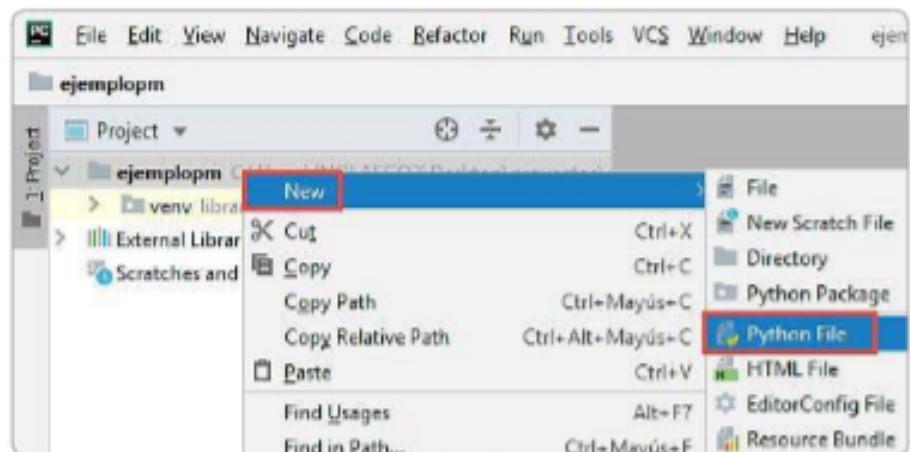
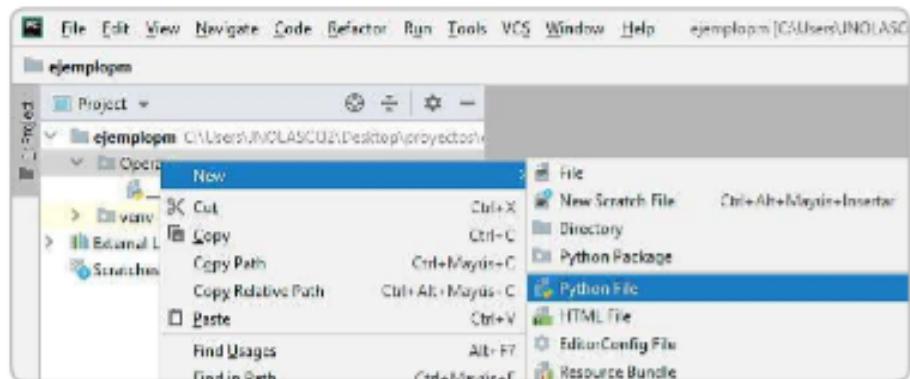
def suma(numero1,numero2):
    return numero1+numero2

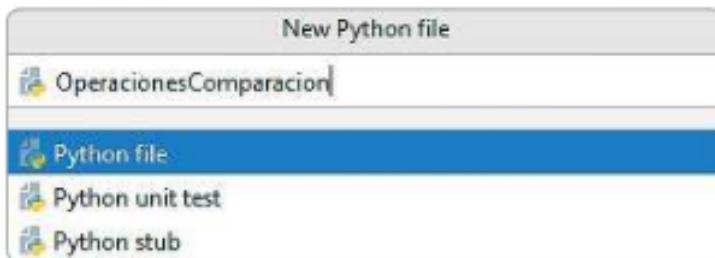
def resta(numero1,numero2):
    return numero1-numero2

def multiplicacion(numero1,numero2):
    return numero1*numero2

def division(numero1,numero2):
    return numero1/numero2
```

Posteriormente, se creará el módulo **OperacionesComparacion.py** dentro del paquete **Operaciones**:





OperacionesComparacion.py

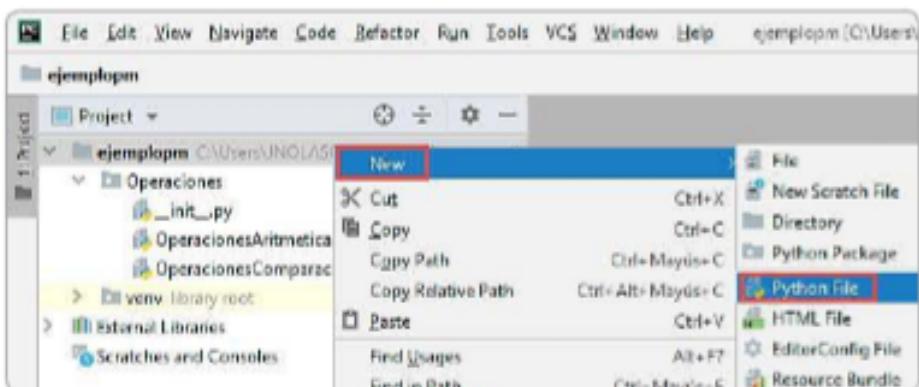
```
#programa :OperacionesComparacion.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#-----#
modulo : OperacionesComparacion
#-----#
def mayor(numero1,numero2):
    if numero1>numero2:
        return numero1
    else:
        return numero2

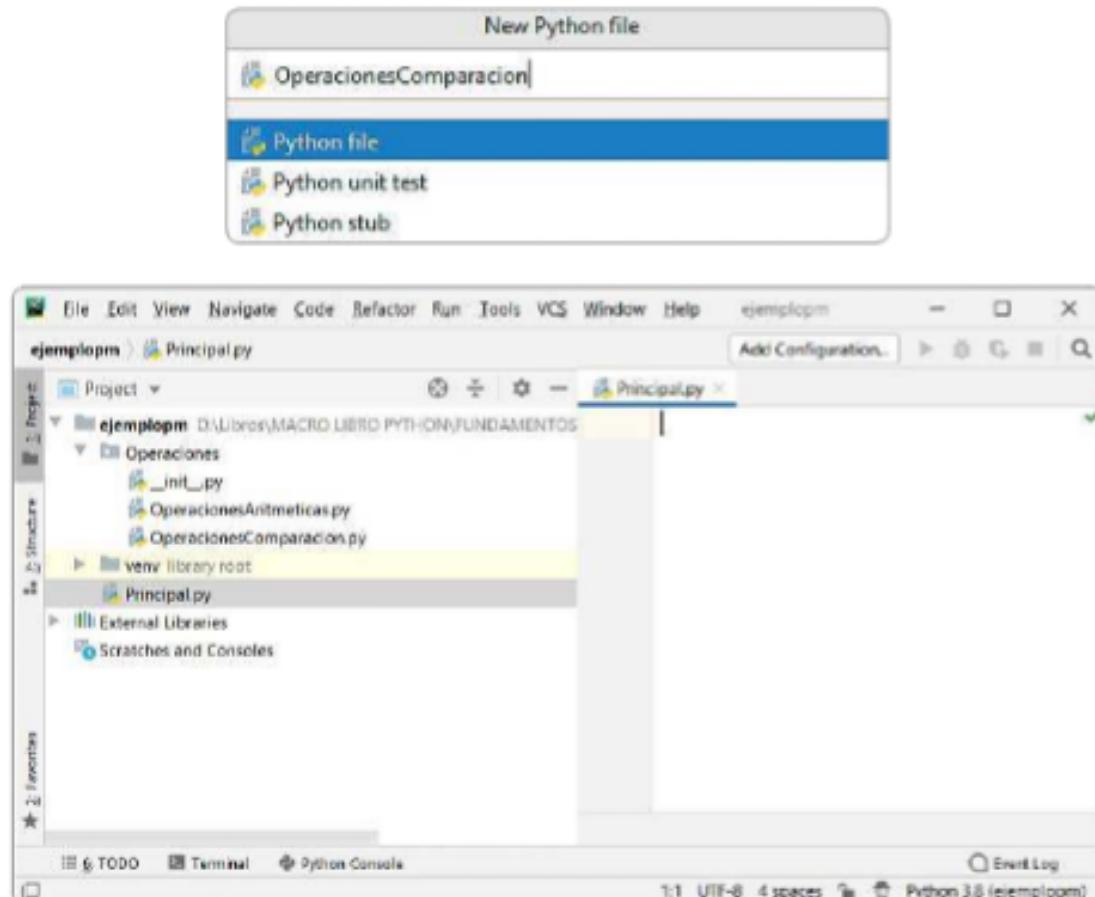
def menor(numero1,numero2):
    if numero1<numero2:
        return numero1
    else:
        return numero2

def comparando(numero1,numero2):
    if numero1==numero2:
        return "Iguales"
    else:
        return "Diferentes"
```

Ahora, se describirá la forma de importar paquetes y módulos en el archivo **Principal.py**:

Principal.py





Para poder acceder al módulo, se puede usar la palabra import. Por ejemplo:

Import modulo1, modulo2, ...modulox

Luego, para acceder a una función o clase:

Modulo1.funcion()

Modulo1.clase()

Asimismo, se puede usar la palabra from. Por ejemplo:

From modulo funcion1, funcion2

A continuación, se demuestran las distintas maneras de acceder a los módulos y funciones:

Acceso	Descripción
import Operaciones	Accediendo al paquete Operaciones
import Operaciones.OperacionesAritmeticas	Accediendo al paquete Operaciones y al modulo OperacionesAritmeticas
import Operaciones.OperacionesComparacion	Accediendo al paquete Operaciones y al modulo OperacionesComparacion

<code>from Operaciones import OperacionesAritmeticas,OperacionesComparacion</code>	Accediendo al paquete Operaciones y a los módulos OperacionesAritmeticas y OperacionesComparacion
<code>from Operaciones.OperacionesAritmeticas import suma</code>	Accediendo al paquete Operaciones y a los módulos OperacionesAritmeticas y función suma

Principal.py

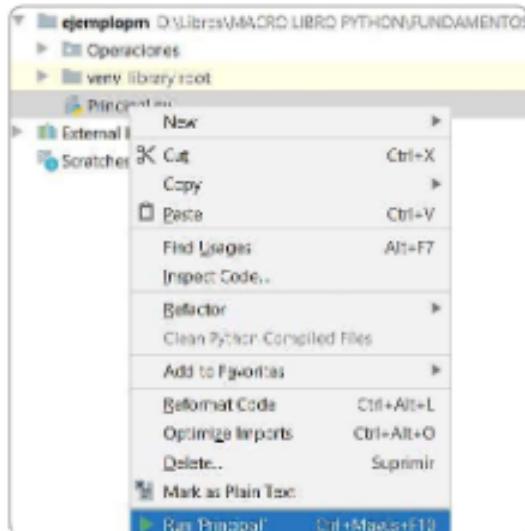
```
#programa : Principal.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de Paquetes y Modulos
"""

accediendo al paquete Operaciones y a
los modulos OperacionesAritmeticas
y todas sus funciones
"""

from Operaciones.OperacionesAritmeticas import *
from Operaciones.OperacionesComparacion import *
if __name__ == '__main__':
    print("Suma====>",suma(1,2))
    print("Resta====>",resta(1,2))
    print("Multiplicacion====>",multiplicacion(1,2))
    print("Division====>,division(1,2))

    print("Mayor====>",mayor(1,2))
    print("Menor====>",menor(1,2))
    print("Comparando====>",comparando(1,2))
```

Posteriormente, se ejecuta y se hace clic derecho sobre el archivo **Principal.py**. Luego, se selecciona **Run "Principal.py"**.



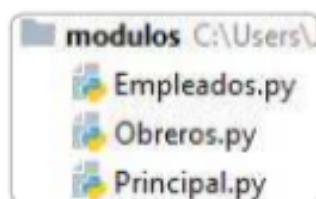
Salida:

```
Suma====> 3
Resta====> -1
Multiplicación====> 2
División====> 0.5
Mayor====> 2
Menor====> 1
Comparando====> Diferentes
```

En un sistema de planillas, se tiene los siguientes módulos:

- Planilla de empleados
- Planilla de obreros

Se crea tres archivos con extensión py:



Luego, se codifica los módulos Empleados y Obreros:

```
Empleados.py
#programa : Empleados.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
modulo : Empleados

def datos(nombres,apellidos,
          cargo,horasm,sueldo):
    print("Su Nombre es :" +nombres)
    print("Su Apellidos son :" +apellidos)
    print("Su cargo es:" +cargo)
    print("Su Horas Mensuales "
          "Trabajadas son :" +horasm)
    print("Su Sueldo es :" +sueldo)

def pagoMensual(horasm,sueldo):
    pagoxhora=sueldo/30/8
    pago=pagoxhora*horasm
    print("Su pago mensual es :"
          +str(pago))
```

Obreros.py

```
#programa : Obreros.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
modulo : Obreros
"""

def datos(nombres,apellidos,
          cargo,horasm,sueldo):
    print("Su Nombre es :" +nombres)
    print("Su Apellidos son :" +apellidos)
    print("Su cargo es:" +cargo)
    print("Su Horas Mensuales"
          " Trabajadas son :" +horasm)
    print("Su Sueldo es :" +sueldo)

def pagoSemanal(horass,salario):
    pagoxhora=salario/7/8
    pago=pagoxhora*horass
    print("Su pago semanal es:"
          +str(pago))
```

Para poder acceder al módulo, se puede usar la palabra import. Por ejemplo:

Import modulo1, modulo2, ...moduloX

Luego, para acceder a una función o clase:

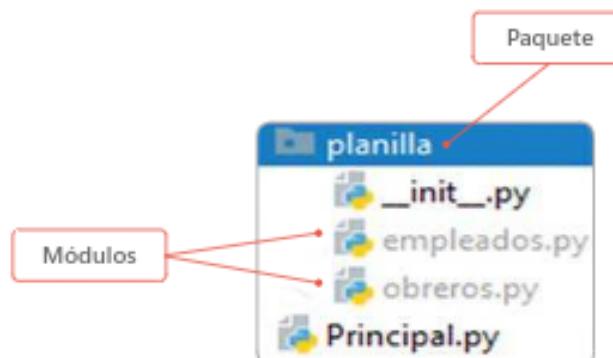
Modulo1.funcion()

Modulo1.clase()

Asimismo, se puede usar la palabra from. Por ejemplo:

From modulo funcion1 , funcion2

A continuación, se crea la carpeta **planilla** y, dentro de esta, se coloca los módulos y un archivo con el nombre de **_init_.py**:



Ahora, se describirá la forma de importar paquetes y módulos en el archivo **Principal.py**:

Principal.py

```
#programa : Principal.py
#autor : jorge nolasco valenzuela
#fecha : 08-08-2017
"""
descripcion : este programa muestra
el uso de Paquetes y Modulos
"""

accediendo al paquete planilla
"""

import planilla
"""

accediendo al paquete planilla
y al modulo empleados
"""

import planilla.empleados
"""

accediendo al paquete planilla y a
los modulos empleados y obreros
"""

from planilla import empleados,obreros
"""

accediendo al paquete planilla y al
modulo empleados y funcion datos
"""

from planilla.empleados import datos
```

Nota

No es recomendable utilizar el * luego del import:

import *. Para utilizar paquetes de terceros use pip:

pip install paquete_de_terceros

Preguntas

Capítulo 2



1. ¿Cuál es el código para identificar el mayor de dos números ingresados? Escribalo.

2. Escriba el código para determinar si una persona pagará impuestos cuando sus ingresos sean superiores a 29 400.

3. ¿Cuál es la salida del siguiente fragmento de código?

```
x = 10

if x == 10:
    print(x == 10)
if x > 5:
    print(x > 5)
if x < 10:
    print(x < 10)
else:
    print("else")
```

4. ¿Cuál es la salida del siguiente fragmento de código?

```
x = "1"  
if x == 1:  
    print("uno")  
elif x == "1":  
    if int(x) > 1:  
        print("dos")  
    elif int(x) < 1:  
        print("tres")  
    else:  
        print("cuatro")  
if int(x) == 1:  
    print("cinco")  
else:  
    print("seis")
```

5. ¿Cuál es la salida del siguiente fragmento de código?

```
x = 1  
y = 1.0  
z = "1"  
  
if x == y:  
    print("uno")  
if y == int(z):  
    print("dos")  
elif x == y:  
    print("tres")  
else:  
    print("cuatro")
```

6. Escriba el código utilizando el ciclo while:

- Pedirá al usuario que ingrese un número entero.
- Utilizará un ciclo while.
- Comprobará si el número ingresado por el usuario es el mismo que el número escogido por usted.
- Si el número elegido por el usuario es diferente al número elegido por usted, el usuario observará el mensaje "continúa en el ciclo" y se le solicitará que ingrese un número nuevamente.

- Si el número ingresado por el usuario coincide con su número, el usuario observará el mensaje "felicidades saliste del ciclo".

7. Escriba el código utilizando el bucle for que cuente de 0 a 20, e imprima números impares en la pantalla.

8. ¿Cuál es la salida del siguiente fragmento de código?

```
for i in range(0, 12, 6):  
    print(i)
```

9. ¿Cuál es la salida del siguiente fragmento de código?

```
x = 5  
y = 8  
e = x >> 2  
f = y << 2  
print(e, f)
```

10. Una función es definida con la palabra clave:

- () function
- () fun
- () def

11. ¿Cuál de las siguientes afirmaciones es falsa?

- () La palabra return devuelve un valor
- () La palabra return termina
- () La palabra return reinicia la ejecución

3

Listas, tuplas, diccionarios, conjuntos y excepciones

3.1 Listas

En muchas ocasiones, se necesita manipular grandes cantidades de valores. Para lograrlo, se tendrá que declarar muchas variables en un tiempo prolongado.

```
Numero1 = int(input("Numero1 :"))
Numero2 = int(input("Numero2 :"))
Numero3 = int(input("Numero3 :"))
Numero4 = int(input("Numero4 :"))
:
:
Numeron = int(input("Numeron :"))
```

En Python, se debe utilizar la estructura denominada "listas".

Sintaxis:

Creación de una lista: [Valor1, Valor2,...]
Acceso a elementos: Lista[Indice]

Creación de listas:

ListaVacia[]
ListaconElementos[11,22,33,44]

Las listas son uno de los tipos de colección que posee el lenguaje Python. Estas son algunas de sus características:

- Son dinámicas, es decir, pueden aumentar y disminuir su tamaño.
- Las listas se crean con dos corchetes, donde se colocan a los elementos separados por comas (,). Los elementos pueden ser de cualquier tipo, incluso listas:

Ejemplo: lista = [13, "xyz", 95.8, 4+7j, [True, "duck"]]

- Pueden concatenarse con otras listas, usando el operador "+", y multiplicar su tamaño:

l = [1, 2, 3]

g = [5, 7, 2]

h = l + g # [1,2,3,5,7,2]

k = g*2 # [5, 7, 2, 5, 7, 2]

- Son objetos mutables, es decir, es posible acceder a sus elementos y modificarlos o borrarlos con ****. Cada índice de la lista hace referencia a un elemento, empezando por el índice 0 que refiere al primero.

```
li = ["b", 4, 10, "c"]
```

```
print li[1] # muestra 4 en pantalla
```

```
del li[2] #[1, 4, "c"] se borró el tercer elemento
```

3.1.1 Posición de elementos

```
lista1 = ["jorge", "pedro", 20, 30]
          0       1       2       3
```

3.1.2 La función Len

Una lista tiene una longitud que puede variar al agregar o eliminar nuevos elementos. Para ello, se debe utilizar la función `Len()`. El nombre de esta función proviene de la longitud.

La función `Len` toma el nombre de la lista como argumento y devuelve el número de elementos actualmente almacenados dentro esta.

A continuación, se creará un programa que demuestre el uso de listas y su función `Len`:

Listas1.py

```
#programa : Listas1.py
#autor : jorge nolasco valenzuela
#fecha : 1-05-2020
#####
descripcion : este programa muestra
el uso de listas - creacion de una Lista
con algunos nombres
#####
lista= ["jorge","pedro","jaime","rosa"]
#numero de elementos de la lista
print("Numero de elementos de la lista :",end="")
print(len(lista))
#muestra los elementos de la lista
for elemento in lista:
    print(elemento)
```

Numero de elementos de la lista :4

```
jorge
pedro
jaime
rosa
```

Luego, se creará un programa que demuestre la suma de elementos de dos listas:

Listas2.py

```
#programa : Listas2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de listas - suma de elementos
de dos listas
"""

lista1 = ["jorge","pedro","jaime","rosa"]
lista2 = ["ana","eva","rosa","jose"]
#muestra el numero de elementos de la lista1
print("Numero de elementos de la lista1 : ",end="")
print(len(lista1))
#muestra los elementos de la lista1
for elemento in lista1:
    print(elemento)
print("====")
# muestra el numero de elementos de la lista2
print("Numero de elementos de la lista2 : ",end="")
print(len(lista2))
# muestra los elementos de la lista2
for elemento in lista2:
    print(elemento)
print("====")
# sumar los elementos de la lista1 y lista2
lista3=lista1+lista2
# muestra el numero de elementos de la lista2
print("Numero de elementos de la lista3 : ",end="")
print(len(lista3))
# muestra los elementos de la lista3
for elemento in lista3:
    print(elemento)
```

Numero de elementos de la lista1 :4

```
jorge
pedro
jaime
rosa
=====
```

Numero de elementos de la lista2 :4

```
ana
eva
rosa
jose
=====
```

Numero de elementos de la lista3 :8

```
jorge
pedro
```

```
jaime
rosa
ana
eva
rosa
jose
```

Posteriormente, se creará un programa que busque un elemento específico en una lista:

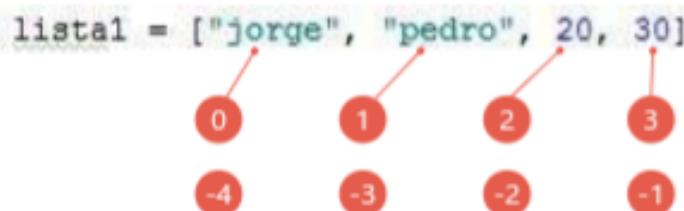
Listas3.py

```
#programa : Listas3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de listas - búsqueda de elemento
"""

lista= ["jorge","pedro","jaime","rosa"]
#preguntar si existe un elemento en la lista
if "rosa" in lista:
    print("encontrado")
else:
    print("no encontrado")
encontrado
```

3.1.3 Índices negativos

Lista [-1] hace mención al último elemento de la lista; lista [-2], al penúltimo; y, así sucesivamente.



3.1.4 Cambiar elementos a una lista

A continuación, se creará un programa que modifique el elemento de una lista:

Listas4.py

```
#programa : Listas4.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de listas - añadir elementos
"""
```

```

lista= ["jorge","pedro","jaime","rosa"]
#añadir elementos
lista[0] = "jose"
# muestra la lista
print(lista)
['jorge', 'pedro', 'jaime', 'rosa', 'alberto', 'luz']

```

3.1.5 Añadir elementos a una lista

A continuación, se creará un programa que añada elementos a una lista:

```

Listas5.py
#programa : Listas5.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripcion : este programa muestra
el uso de listas - añadir elementos
"""

lista= ["jorge","pedro","jaime","rosa"]
#añadir elementos
lista.append("alberto")
lista.append("luz")
# muestra la lista
print(lista)
['jorge', 'pedro', 'jaime', 'rosa', 'alberto', 'luz']

```

Nota



El método `insert()` también puede ser utilizado para agregar un nuevo elemento en cualquier lugar de la lista:

`list.insert(localizacion,valor)`

3.1.6 Eliminar elementos a una lista

```

Listas6.py
#programa : Listas6.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripcion : este programa muestra
el uso de listas - eliminar elementos
"""

lista= ["jorge","pedro","jaime","rosa"]
#añadir elementos
lista.remove("pedro")
#imprimir la lista
print(lista)
['jorge', 'jaime', 'rosa']

```

3.1.7 Algunos ejemplos de listas

Lista7.py

```
#programa : Lista7.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#####
descripcion : este programa muestra
el uso de Listas - tabla ascii
#####

#lista tascii
tascii=[]
columnas=11
for numero in range(32,256):
    #adicionar a la lista tascii los elementos
    tascii.append(chr(numero))
print("=*10," Tabla Ascii ",=*10)
print("=*35)
#mostrar los elementos de la lista tascii
for ele in tascii:
    if columnas>0:
        print(ele+" ",end="")
        columnas = columnas - 1
    elif columnas==0:
        print(ele)
        columnas = 11
#salto de linea
print("\n")
print("=* * 35)
```

Nota

Es conveniente mencionar algunas funciones importantes:

`chr(i)` devuelve una cadena de un carácter cuyo código ASCII es el entero *i*.

`ord(c)` devuelve el valor ASCII de una cadena de un carácter o un carácter Unicode.

3.1.8 Intercambiar elementos de una lista

También es posible realizar intercambio de elementos de una lista.

A continuación, se muestra un ejemplo del intercambio entre el primer y último elemento de una lista:

intercambio.py

```
#programa : intercambio.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripcion : este programa muestra
el intercambio de valores de una Lista
"""

Lista = [22, 33, 48, 1, 0]
intercambio = True
Lista[0], Lista[4] = Lista[4], Lista[0]
print(Lista)
```

[0, 33, 48, 1, 22]

3.1.9 Ordenamiento de listas

Ahora, se observará cómo ordenar los elementos de una lista. Muchos algoritmos de clasificación se han inventado hasta ahora, que difieren mucho en velocidad, así como en complejidad.

ordenamientoListas1.py

```
#programa : OrdenarLista1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripcion : este programa muestra
el ordenamiento de una Lista
"""

Lista = [22, 33, 48, 1, 0]
intercambio = True
while intercambio:
    intercambio = False
    for i in range(len(Lista) - 1):
        if Lista[i] > Lista[i + 1]:
            intercambio = True
            Lista[i], Lista[i + 1] = Lista[i + 1], Lista[i]
print(Lista)
```

[0, 1, 22, 33, 48]

ordenamientoListas2.py

```
#programa : OrdenarLista2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#MM

descripcion : este programa muestra
el ordenamiento de una Lista
#MM

Lista = []
intercambio = True
numero = int(input("Número de Elementos de la Lista:"))

for elemento in range(numero):
    val = float(input("Ingrese Elemento de la Lista: [" + str(elemento) + "]:"))
    Lista.append(val)

while intercambio:
    intercambio = False
    for i in range(len(Lista) - 1):
        if Lista[i] > Lista[i + 1]:
            intercambio = True
            Lista[i], Lista[i + 1] = Lista[i + 1], Lista[i]
print(Lista)
[0, 1, 22, 33, 48]
```

3.1.10 Método sort()

Se puede usar el método `sort()` para ordenar elementos de una lista, por ejemplo:

Sort1.py

```
#programa : sort1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#MM

descripcion : este programa muestra
el ordenamiento mediante el
método sort
#MM

Lista = [15, 155, 5, 1, 200]
print("Antes del ordenamiento")
print(Lista)
Lista.sort()
print("Después del ordenamiento")
print(Lista)
```

Antes del ordenamiento

[15, 155, 5, 1, 200]

Después del ordenamiento

[1, 5, 15, 155, 200]



3.1.11 Método reverse()

También hay un método de lista llamado `reverse()`, que se puede usar para invertir la lista. Por ejemplo:

reverse1.py

```
#programa : reverse1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#versión 1.0

descripcion : este programa muestra
el uso del metodo reverse
para invertir valores

lista = [100, 99, 98, 97, 96]
print("Antes de Invertir")
print(lista)
print("Invertir Lista")
lista.reverse()
print(lista)
```

Antes de Invertir

[100, 99, 98, 97, 96]

Invertir Lista

[96, 97, 98, 99, 100]

3.1.12 Limitar los elementos de una lista

limitar1.py

```
#programa : limitar1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#versión 1.0

descripcion : este programa muestra
como limitar los valores de una lista

Lista = [50, 40, 30, 60, 90]
Lista2 = Lista[1:3]
print(Lista2)

[40, 30]
```

Nota

La nueva lista (lista 2) tendrá $3 - 1 = 2$ elementos.

Los números elementos son aquellos cuyo índice son 1 y 2.



limitar2.py

```
#programa : limitar2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripción : este programa muestra
cómo limitar los valores de una lista
"""


```

```
List = [10, 8, 6, 4, 2]
```

```
Lista2 = List[1:-2]
```

```
print(Lista2)
```

```
[8, 6]
```

Nota

La nueva lista comienza en el elemento 1 (8), y no incluye elementos -1 y -2 que son 4 y 2.

limitar3.py

```
#programa : limitar3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripción : este programa muestra
cómo limitar los valores de una lista
"""


```

```
List = [50, 40, 30, 60, 90]
```

```
Lista2 = List[:2]
```

```
print(Lista2)
```

```
[50, 40]
```

Nota

La nueva lista solo incluye los dos primeros elementos.

3.1.13 Listas en listas

Las listas pueden consistir en números y elementos de una estructura mucho más compleja.

A continuación, se muestran algunos ejemplos:

Llist1.py

```
#programa : Llist1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripción : este programa muestra
cómo utilizar Listas mas complejas
"""


```

```
lista1 = [elemento ** 2 for elemento in range(10)]
print(lista1)
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

Llista2.py

```
#programa : Llist2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
como utilizar Listas mas complejas
"""

lista1 = [2 ** elemento for elemento in range(10)]
print(lista1)
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512]
```

LlistaPares.py

```
#programa : LlistaPares.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
como utilizar Listas complejas
para mostrar elementos pares
"""

lista1 = [elemento**1 for elemento in range(10)]
ListaPares = [elemento for elemento in lista1 if elemento%2==0]
print(ListaPares)
[[0, 2, 4, 6, 8]]
```

LlistalImpares.py

```
#programa : LlistalImpares.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
como utilizar Listas complejas
Numero Impares

"""

lista1 = [elemento**1 for elemento in range(10)]
ListalImpares = [elemento for elemento in lista1 if elemento%2!=0]
print(ListalImpares)
[1, 3, 5, 7, 9]
```

3.1.14 Listas bidimensionales

En algunos casos, es necesario crear listas bidimensionales para almacenar datos en listas rectangulares. Tales tablas se llaman matrices o matrices bidimensionales.

Por ejemplo, si se desea mostrar las tres calificaciones de 3 alumnos.

A continuación, se muestran algunos ejemplos:

ListaB1.py

```
#programa : ListaB1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
como utilizar Listas bidimensionales
"""

calificaciones = [[11, 12, 20], [11, 12, 20], [11, 12, 20]]
print("Calificacion del Alumno1====>",end="")
print(calificaciones[0][0],end=",")
print(calificaciones[0][1],end=",")
print(calificaciones[0][2])
print("Calificacion del Alumno2====>",end="")
print(calificaciones[1][0],end=",")
print(calificaciones[1][1],end=",")
print(calificaciones[1][2])
print("Calificacion del Alumno3====>",end="")
print(calificaciones[2][0],end=",")
print(calificaciones[2][1],end=",")
print(calificaciones[2][2])
```

```
[Calificacion del Alumno1====>11,12,20
Calificacion del Alumno2====>11,12,20
Calificacion del Alumno3====>11,12,20
```

ListaB2.py

```
#programa : ListaB2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
como utilizar Listas bidimensionales
"""

calificaciones = [[11, 12, 20], [11, 12, 20], [11, 12, 20]]
for filas in calificaciones:
    for elemento in filas:
        print(elemento,end=",")
    print()
```

```
11,12,20,
11,12,20,
11,12,20,
```

3.2 Tuplas

Las tuplas son otra colección de Python muy similar a las listas, con la diferencia que estas son inmutables. Sus características son las siguientes:

- Las tuplas se crean con dos paréntesis, donde se colocan los elementos separados con comas. Estos pueden ser de cualquier tipo incluso tuplas: t=(23,45,("abc","efg")).
- t[0]=6 #TypeError: "tuple" object does not support item assignment.
- No son dinámicas, es decir, una vez creadas no pueden aumentar o disminuir la cantidad de elementos del t[1] #TypeError: "tuple" object doesn't support item deletion.

3.2.1 Ejemplos

A continuación, se creará un programa que demuestre el uso de tuplas:

Tupla1.py

```
#programa : Tupla1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#####
descripcion : este programa muestra
el uso de Tuplas - creacion de una Tupla
con nombres
#####
tupla1= ("jorge","pedro","jaime","rosa")
#numero de elementos de la tupla
print("Numero de elementos de la tupla:",end="")
print(len(tupla1))
# muestra los elementos de la tupla
for elemento in tupla1:
    print(elemento)
```

Numero de elementos de la tupla:4

jorge
pedro
jaime
rosa

Ahora, se creará un programa que demuestre el uso de concatenación y anidación de tuplas:

Tupla2.py

```
#programa : Tupla2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#####
descripcion : este programa muestra
el uso de Tuplas - concatenar y anidar Tuplas
#####
alumno1= ("Luis","av. xyz",20)
alumno2= ("Eva","av. abc",30)
```

```
#concatenar tupla
nuevatupla1=alumno1+alumno2
#anidar tupla
nuevatupla2=alumno1,alumno2
# muestra las tuplas
print(nuevatupla1)
print(nuevatupla2)

('Luis', 'av. xyz', 20, 'Eva', 'av. abc', 30)
 (('Luis', 'av. xyz', 20), ('Eva', 'av. abc', 30))
```

3.2.2 Creación de tuplas vacías

Es posible crear tuplas vacías de la siguiente manera:

```
TuplaVacia=()
```

3.2.3 Mostrar elementos de una tupla

Es posible mostrar elementos de una tupla de la siguiente manera:

Tupla3.py

```
#programa : Tupla3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#####
descripción : este programa muestra
elementos de una Tupla
#####

Tupla1 = (100, 200, 300, 400, 500)
#Imprime todo los elementos de la Tupla
for elemento in Tupla1:
    print(elemento)
#Imprime el primer elemento de la Tupla
print(Tupla1[0])
#Imprime el ultimo elemento de la Tupla
print(Tupla1[-1])
#Imprime desde el segundo elemento de la Tupla
print(Tupla1[1:])
#Imprime los tres primeros elementos de la Tupla
print(Tupla1[:-2])
```

```
100
200
300
400
500
100
500
(200, 300, 400, 500)
(100, 200, 300)
```



No intente modificar el contenido de una tupla:

```
Tupla1 = (100, 200, 300, 400, 500)
```

```
Tupla1.append(600)
```

```
AttributeError: "tuple" object has no attribute "append".
```

3.3 Diccionarios

Los diccionarios son colecciones, en las cuales se asocia explícitamente un conjunto de índices a un conjunto de datos, llamados "llave: valor". Estos son similares a los arrays asociativos en otros lenguajes como PHP. Los pares dentro del diccionario no tienen un orden establecido.

Estas son sus características:

- Se definen con llaves {}. Dentro de estos pares de datos (de la forma key: value), las llaves deben ser objetos inmutables (int, str, tuple, etc):

```
diccionario = {"Silvia": 29, "Pepe":32, 100:"total"}
```
- Se puede acceder a los elementos del diccionario a través de sus llaves.
- Es posible modificar los valores de la misma manera que con las listas.
- Para agregar un nuevo par de elementos al diccionario, se realiza el mismo procedimiento que cuando se modifica un elemento como si ya existiese.
- Para eliminar un par de datos del diccionario, se puede aplicar la sentencia :

```
print diccionario["Silvia"]    #muestra 29 en pantalla
diccionario["Silvia"] = 27    #{"Silvia":27, "Pepe":32, 100:"total"}
diccionario["Ana"] = 5        #{"Silvia":27, "Pepe":32, "Ana":5, 100:"total"}
del diccionario["Silvia"]    #{"Pepe":32, 100:"total"}
```

temperaturas =	Clave	Valor
{"Dubai":31,	Dubai	31
"Bruselas":15,	Bruselas	15
"Zurich":17,	Zurich	17
"Hamburgo":16,	Hamburgo	16
"Paris":17,	Paris	17
"Roma":25,	Roma	25
"Shanghai":40,	Shanghai	40
"Munich":17,	Munich	17
"Londres":14}	Londres	14

temperaturas [0]	31
	15
	17
	16
temperaturas ["Paris"]	17
	25
	40
	17
temperaturas [8]	14

3.3.1 Métodos de los diccionarios

Declaración de un diccionario.

```
temperaturas = {"Dubai":31,
                 "Bruselas":15,
                 "Zurich":17,
                 "Hamburgo":16,
                 "Paris":17,
                 "Roma":25,
                 "Shanghai":40,
                 "Munich":17,
                 "Londres":14}
```

Número de elementos que tiene el diccionario.

```
len(temperaturas)
```

Devolver una lista con las claves del diccionario.

```
temperaturas.keys()
```

Devolver una lista con los valores del diccionario.

```
temperaturas.values()
```

Se inserta un elemento en el diccionario con su clave: valor.

```
temperaturas ["Lima"] = 20
```

Se elimina el elemento del diccionario con clave key.

```
temperaturas.pop("Lima",None)
```

3.3.2 Ejemplos

A continuación, se crearán algunos programas que demuestren el uso del diccionario:

Diccionario1.py

```
#programa : Diccionario1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de Diccionarios - Telefonos de personas
"""

#crea el diccionario telefono
telefono = {"Juan":512645011,"Pedro":512644099,"jaime":512611099}
#muestra los elementos del diccionario telefono
print(telefono)
#mostrar el telefono de Juan
print(telefono["Juan"])

{'Juan': 512645011, 'Pedro': 512644099, 'jaime': 512611099}
512645011
```

Diccionario2.py

```
#programa : Diccionario2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de Diccionarios - Telefonos de personas
"""

#crea el diccionario telefono
telefono = {"Juan":512645011,"Pedro":512644099,"jaime":512611099}
#muestra los elementos del diccionario telefono
for key in telefono.keys():
    print(key, "->", telefono[key])

Juan -> 512645011
Pedro -> 512644099
jaime -> 512611099
```

3.3.3 Función sorted()

La función sorted() se utiliza para ordenar diccionarios.

A continuación, se creará un ejemplo:

Diccionario3.py

```
#programa : Diccionario3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
```

descripcion : este programa muestra el uso de la función sorted para ordenar elementos de un diccionario

```
# crea el diccionario telefono
telefono = {2:"luis",1:"Pedro",3:"jaime"}
# muestra los elementos del diccionario telefono ordenadamente
for key in sorted(telefono.keys()):
    print(key, "->", telefono[key])
```

1 -> Pedro

2 -> luis

3 -> jaime

3.3.4 Utilizar los métodos item() y values()

Otra manera de listar diccionarios es usar el método `item()`. Este método devuelve una lista de tuplas, donde cada tupla es un par clave:valor.

Ahora, se creará un ejemplo:

Diccionario4.py

```
# programa : Diccionario4.py
# autor : jorge nolasco valenzuela
# fecha : 01-05-2020
# descripcion : este programa muestra el uso del método items para listar los elementos de un diccionario
```

```
# crea el diccionario personas
personas = {19668210:"luis",19668211:"Pedro",19668212:"jaime"}
# muestra los elementos del diccionario personas
for dni, nombre in personas.items():
    print(dni, "->", nombre)
```

19668210 -> luis

19668211 -> Pedro

19668212 -> jaime

También se podría utilizar el método `values()`.

A continuación, se creará un ejemplo:

Diccionario5.py

```
# programa : Diccionario5.py
# autor : jorge nolasco valenzuela
# fecha : 01-05-2020
# descripcion : este programa muestra el uso del método values para listar los elementos de un diccionario
```

```
#crea el diccionario personas
personas = {19668210:"luis",19668211:"Pedro",19668212:"jaime"}
#muestra los elementos del diccionario personas
for nombre in personas.values():
    print(nombre)
```

**luis
Pedro
jaime**

3.3.5 Modificar, ingresar y eliminar valores de un diccionario

Ahora, se creará un programa en el cual se ingresarán los elementos de un diccionario:

```
#modificar
personas = {19668210:"luis",19668211:"Pedro",19668212:"jaime"}
dict[19668210] = 'jose'
```

```
#Ingresar
personas = {19668210:"luis",19668211:"Pedro",19668212:"jaime"}
dict[19668213] = 'eva'
```

```
#Eliminar
personas = {19668210:"luis",19668211:"Pedro",19668212:"jaime"}
del dict[19668210]
```

Nota



Puede eliminar el último elemento en un diccionario con el método `popitem()`.

3.3.6 Ingresar elementos a un diccionario

A continuación, se creará un programa en el cual se ingresarán los elementos de un diccionario:

```
Diccionario6.py
#programa : Diccionario6.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#####
descripcion : este programa muestra
el uso de Diccionarios - Ingresando Telefonos de personas
#####
#crea el diccionario telefono
telefonos = {}
while True:
    print("=====SELECCIONE OPCION=====")
    print("1: INGRESAR")
    print("2: LISTAR")
    print("4: SALIR")
```

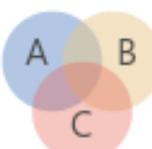
```

opc=int(input("SELECCIONE OPCION :"))
if opc==1:
    nom = input("Ingrese Nombre :")
    tel = int(input("Ingrese Telefono :"))
    telefonos[nom] = tel
elif opc==2:
    # muestra los elementos del diccionario
    print(telefonos)
else:
    break
=====SELECCIONE OPCION=====
1: INGRESAR
2: LISTAR
4: SALIR
SELECCIONE OPCION :1
Ingrese Nombre :jorge
Ingrese Telefono :5199966103
=====SELECCIONE OPCION=====
1: INGRESAR
2: LISTAR
4: SALIR
SELECCIONE OPCION :2
{'jorge': 5199966103}
=====SELECCIONE OPCION=====
1: INGRESAR
2: LISTAR
4: SALIR
SELECCIONE OPCION :3

```

3.4 Conjuntos

En Python, un conjunto es una colección no ordenada. Sus elementos no son repetidos y sus usos básicos contienen verificación de pertenencia y eliminación de entradas duplicadas. Los conjuntos también soportan operaciones matemáticas. Así, por ejemplo, la unión, la intersección, la diferencia, y la diferencia simétrica.



A continuación, se crea un conjunto denominado "frutas":

```
frutas = {"platano", "manzana", "naranja", "manzana", "pera", "naranja"}
```

Se muestran los elementos:

```
print(frutas)
{'naranja', 'platano', 'manzana', 'pera'}
```

Se verifica si el elemento pertenece al conjunto.

```
if "platano" in frutas:
    print("Pertenece al conjunto")
else:
    print("No Pertenece al conjunto")
```

3.4.1 Ejemplos

Unión de conjuntos:

Conjunto1.py

```
#programa : Conjunto1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#####
descripcion : este programa muestra
el uso basico de Conjuntos - union
#####

#Creamos 4 conjuntos
conjunto1={1,2,3,4,5}
conjunto2={6,7,8,9,10}
conjunto3={11,12,13,14}
conjunto4={1,2,3}
conjunto5={}
#union de dos conjuntos
conjunto5 = conjunto1 | conjunto2
print(conjunto5)
#union de tres conjuntos
conjunto6 = conjunto1 | conjunto2 | conjunto3
print(conjunto6)

{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14}
```

Intersección de conjuntos:

Conjunto2.py

```
#programa : Conjunto2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#####
descripcion : este programa muestra
el uso basico de Conjuntos - intersección
#####

#Creamos 3 conjuntos
conjunto1={1,2,3,4,5}
conjunto2={1,2,3}
conjunto3={}
#intersección de dos conjuntos
conjunto3 = conjunto1 & conjunto2
print(conjunto3)

{1, 2, 3}
```

Diferencia de conjuntos:

Conjunto3.py

```
#programa : Conjunto3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#####
descripcion : este programa muestra
el uso basico de Conjuntos - diferencia
#####
#Creamos 3 conjuntos
conjunto1={1,2,3,4,5}
conjunto2={5,6,7,8,9}
conjunto3={}
#diferencia de dos conjuntos
conjunto3 = conjunto1 - conjunto2
print(conjunto3)
```

{1, 2, 3, 4}

Unión de exclusiva de conjuntos:

Conjunto4.py

```
#programa : Conjunto4.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#####
descripcion : este programa muestra
el uso basico de Conjuntos - union
#####
#Creamos 4 conjuntos
conjunto1={1,2,3,4,5}
conjunto2={4,5,6,7,8}
conjunto3={}
#union simetrica de dos conjuntos
conjunto3 = conjunto1^conjunto2
print(conjunto3)
```

{1, 2, 3, 6, 7, 8}

Nota



Se puede utilizar los siguientes métodos para unión e intersección:

Conjunto1.union(conjunto2)
 Conjunto1.interseccion(conjunto2)
 conjunto1.difference(conjunto2)
 conjunto1.symmetric_difference(conjunto2)

3.5 Excepciones

Los errores de ejecución son llamados en general excepciones. Durante la ejecución de un programa, si dentro de una función aparece una excepción y la función no la maneja, la excepción se propaga hacia la función que la invocó.

Python posee un manejo de excepciones avanzado para detectar eventos y modificar el flujo de la ejecución del programa.

A continuación, se muestra un ejemplo de manejo de excepciones:

Ejemplo0.py

```
#programa : Ejemplo0.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#*****#
descripcion : este programa muestra
el uso de Excepciones
con listas
#*****#
#Creamos una lista con 2 elementos
lista=[1,2]
#intentamos mostrar el cuarto elementos
print(lista[3])
```

Traceback (most recent call last):

```
  File  "C:/Users/JNOLASCO2/Desktop/proyectos/otrosproyectos/Ejemplo0.py", line 12, in
<module>
    print(lista[3])
IndexError: list index out of range
```

Se ha identificado el error `IndexError`. Ahora, se capturada la excepción:

Ejemplo0.py

```
#programa : Ejemplo0.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#*****#
descripcion : este programa muestra
el uso de Excepciones
con listas
#*****#
#Creamos una lista con 2 elementos
lista=[1,2]
#intentamos mostrar el cuarto elementos
try:
    print(lista[3])
except IndexError:
    print("Error Indice No Existe")
```

```

else:
    print("Todo Bien")
finally:
    print("Finalizo la Ejecucion")

```

Error Indice No Existe
Finalizo la Ejecucion

3.5.1 Algunos ejemplos de uso de excepciones

3.5.1.1 Validación de números

Ejemplo1.py

```

#programa : Ejemplo1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de Excepciones
Ingreso de Solo Numeros
"""

while True:
    try:
        x=int(input
              ("Por Favor "
               "Ingrese un Numero :"))
        break
    except ValueError:
        print("Solo Numeros")

```

Por Favor Ingrese un Numero :a
Solo Numeros
Por Favor Ingrese un Numero :x
Solo Numeros
Por Favor Ingrese un Numero :3

3.5.1.2 División por cero

Ejemplo2.py

```

#programa : Ejemplo2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de Excepciones
Ingreso solo de Numeros
y division por cero
"""


```

```

while True:
    try:
        numero1=int(input
            ("Ingrese "
            "Primer Numero:"))
        numero2=int(input
            ("Ingrese"
            " Segundo Numero:"))
        division=numero1/numero2
        print("Division :" +str(division))
        break
    except ValueError:
        print("Solo Numeros")
    except ZeroDivisionError:
        print("Division por Cero")

```

Ingrese Primer Numero:20
 Ingrese Segundo Numero:0
 Division por Cero
 Ingrese Primer Numero:2
 Ingrese Segundo Numero:2
 Division :1.0

3.5.1.3 Lectura de ficheros

Ejemplo3.py

```

#programa : Ejemplo3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
***
descripcion : este programa muestra
el uso de Excepciones
Lectura de Ficheros
***
while True:
    try:
        archivo=open("docuemnto.txt")
        for linea in archivo.readline():
            print(linea)
    except FileNotFoundError:
        print("Archivo no Encontrado")
        break

```

Archivo no Encontrado

» Excepciones personalizadas

En algunas oportunidades, es necesario definir excepciones propias. Para ello, se debe realizar lo siguiente:

Crear una clase:

```
class Error(Exception):
    def __init__(self,valor):
        print("El error es debido a : ", valor)
```

```
numero1=10
numero2=0
try:
    resultado = numero1/numero2
    print("Division : ", resultado)
except Error:
    print("Error Mostrado")
```

```
File "C:/Users/jorge/Desktop/libro python/fuentes libro python/Excepciones/Error.py", line 8,
in <module>
    resultado = numero1/numero2
ZeroDivisionError: division by zero
```

» Intérpretes online

Hay situaciones en las que no se desea instalar Python en su computadora y se necesita herramientas que permitan ejecutar scripts. Existe una lista de sitios web que permiten ejecutar código correcto en el navegador:

1. <https://repl.it/languages>
2. <https://www.sourcelair.com/home>
3. <http://www.pythontutor.com/visualize.html#mode=edit>
4. <http://pythonfiddle.com/>
5. <http://rextester.com/runcode>
6. <http://techmums.co/python.html>
7. <https://www.ideone.com/>

A continuación, se utilizará **repl.it**:

The screenshot shows the repl.it homepage. At the top left is the repl.it logo. To its right is a search bar containing the placeholder text "search for a language, e.g. c++". Above the search bar is a dropdown menu labeled "Popular". Below the search bar, there are two language options: "Python3: A dynamic language emphasizing readability." which is highlighted with a red border, and "Ruby: A natural dynamic object-oriented language."

The screenshot shows the repl.it Python3 Compile window. The title bar says "repl.it - Python3 Compile". The address bar shows a secure connection to "https://repl.it/K88s/1". The main interface includes a file browser, a "Log In" button, and a code editor on the left containing the following Python code:

```
1 #uso de interprete Online
2 print("Bienvenidos \n Python
      \n guia Completa")
```

To the right is a terminal window showing the output of the code execution:

```
Python 3.6.1 (default, Dec 2015,
13:05:11)
[GCC 4.8.2] on linux
>
Bienvenidos
Python
guia Completa
> |
```

Preguntas

Capítulo 3



1. ¿Cuál es el resultado del siguiente fragmento?

```
lista = [1,2,3,4,5,6]
```

```
lista.insert(1,7)
```

```
del lista[5]
```

```
lista.append(8)
```

```
print(lista)
```

2. ¿Cuál es el resultado del siguiente fragmento?

```
lista = []
```

```
del lista
```

```
print(lista)
```

3. ¿Cuál es el resultado del siguiente fragmento?

```
lista = [11, [22,33], 43]
```

```
print(lista[1])
```

```
print(len(lista))
```

4. ¿Cuál es el resultado del siguiente fragmento?

```
Tupla = (1, 2, 3)
```

```
print(Tupla [2])
```

5. ¿Cuál es el resultado del siguiente fragmento?

```
Tupla = 10, 20, 30
```

```
a, b, c = Tupla
```

```
print(a * b * c)
```

6. ¿Cuál es el resultado del siguiente fragmento?

```
diccionario1 = {'Juan Perez Perez':'A', 'Jaime Davila Alva':'B'}
```

```
diccionario2 = {'Jose Alva Alva':'C', 'Jorge Nolasco Valenzuela':'D'}
```

```
diccionario3 = {}
```

```
for item in (diccionario1, diccionario2):
```

```
    diccionario3.update(item)
```

```
print(diccionario3)
```

7. ¿Cuál es el resultado del siguiente fragmento?

```
diccionario1 = {"A":1,"B":2,"C":3,"D":4}  
copiadiccionario1 = diccionario1.copy()  
diccionario1.clear()  
print(diccionario1)  
print(copiadiccionario1)
```

8. ¿Cuál es el resultado del siguiente fragmento?

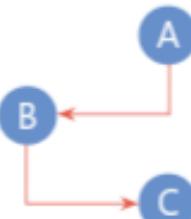
```
colores = {  
    "white" : "blanco",  
    "red" : "rojo",  
    "blue": "azul",  
    "green" : "verde"  
}  
for ing, esp in colores.items():  
    print(ing, "=>", esp)
```


4

Programación orientada a objetos y sus funciones

4.1 Programación orientada a objetos

La programación orientada a objetos es un paradigma. Presenta elementos importantes que son aplicados a los nuevos ambientes de desarrollo y creación de dispositivos portátiles. Esta programación se ha expandido a diferentes lenguajes de programación, adaptándose a los cambios y a las nuevas tendencias tecnológicas.



4.1.1 Introducción a la POO

Se puede observar que estamos rodeados de objetos; por ejemplo, una mesa, una silla, un bolígrafo, etc. Estos podrían clasificarse (con mayor o menor detalle) atendiendo a una definición. Además, a pesar de existir muchos tipos de mesas, estas tienen características comunes. Por lo tanto, es posible concluir que pueden hacerse dos distinciones:

1. El concepto de lo que es una mesa
2. Los tipos de mesas

Esta idea fue trasladada a la informática y surgieron los conceptos de clase y objeto. Se puede decir que una clase es un concepto sobre una entidad abstracta que define cómo serán todos los objetos que existan de ese tipo.

Si se traslada la definición anterior, se puede decir que una clase es un prototipo que define las propiedades y los métodos comunes a múltiples objetos de un mismo tipo. Sería como una plantilla para la creación de objetos. Por otra parte, un objeto es un conjunto de propiedades y métodos capaces de manipular dichas propiedades.

4.1.2 Definición de clase

La palabra "clase" es como una categoría o un resultado de similitudes definidas con precisión.

```
class Persona:  
    pass
```

Nombre de clase

Atributos

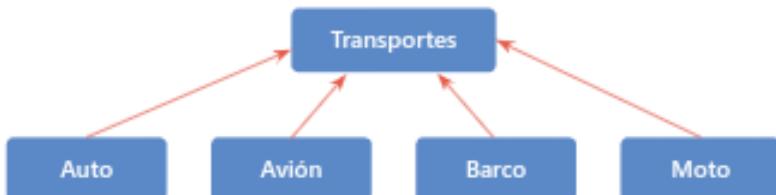
Métodos



El nombre de las clases se define en singular, utilizando CamelCase.

La palabra "clase" es como una categoría o un resultado de similitudes definidas con precisión.

Se intentará señalar algunas clases que son buenos ejemplos de este concepto.



Se observa que todos estos transportes existen y están relacionados por una característica fundamental: pueden moverse, pero son diferentes a todo aquello ser que se mueve.

Nota

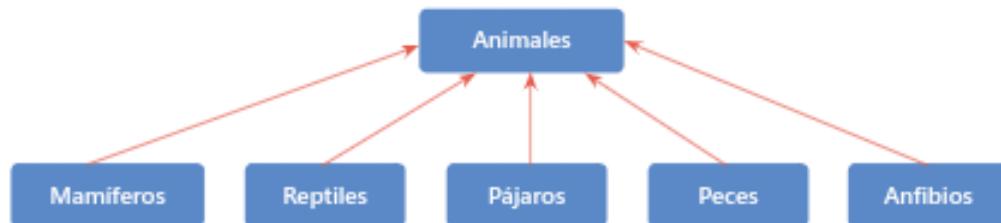


La jerarquía se da de arriba hacia abajo. Desde las clases más generales a las más especializadas, se encuentra siempre en la parte superior (la superclase), mientras que sus descendientes se encuentran debajo (las subclases).

Ahora, se define algunas subclases para la superclase transportes:

- Auto
- Avión
- Barco
- Moto

Además, se describe otro ejemplo: el reino animal. Se puede decir que todos los animales (de clase de nivel superior) se pueden dividir en cinco subclases:



Las clases son las plantillas para hacer objetos. Una clase sirve para definir una serie de objetos con propiedades (atributos), comportamientos (operaciones o métodos), y semántica común.

En las clases se definen sus siguientes aspectos:

- **Atributos.** Es decir, los datos miembros de una clase. Estos datos pueden ser públicos (accesibles desde otra clase) o privados (solo accesibles por código de su propia clase). También se las llama "campos".
- **Métodos.** Es decir, las funciones miembros de la clase. Son las acciones (u operaciones) que puede realizar la clase.
- **Código de inicialización.** Para crear una clase, hace falta realizar normalmente operaciones previas. Esto es conocido como "el constructor de la clase".

4.1.3 Definición de objetos

Una clase es un conjunto de objetos. Por lo tanto, un objeto pertenece a una clase; además, es una especie de copia de una clase con todas sus características y rasgos similares.

4.1.4 Herencia

La herencia permite crear una clase general primera (superclase) y luego crear clases más especializadas que reutilicen el código de la clase general. La herencia también permite escribir un código más limpio y legible.

Por ejemplo, se define una clase general: persona que posee las siguientes características:

Documento de identidad

Nombres

Apellidos

Dirección

:

:

Y también sus métodos:

Hablar

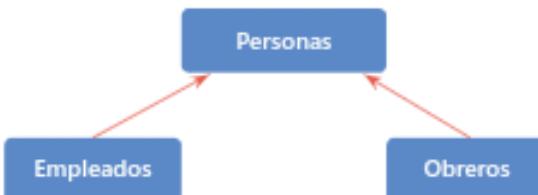
Comer

Caminar

:

:

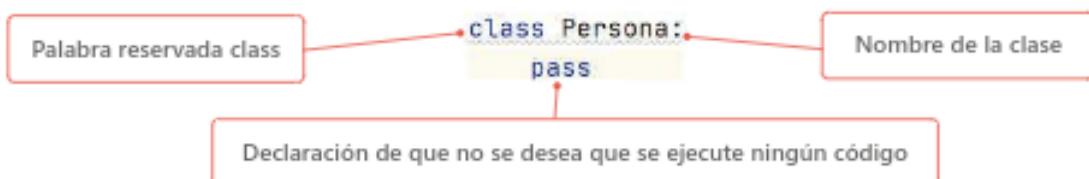
A "empleados" y "obreros" se les puede denominar como sus "clases hijas", ya que poseen características similares.



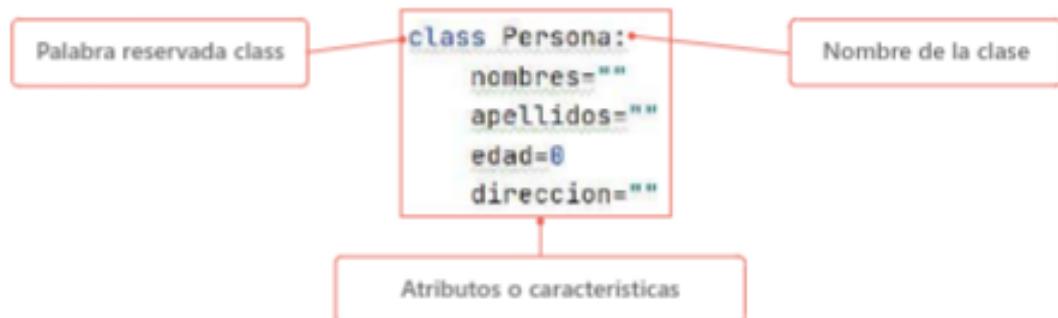
4.1.5 Nuestra primera clase

A continuación, se muestra la clase "persona" con sus respectivos atributos y métodos:

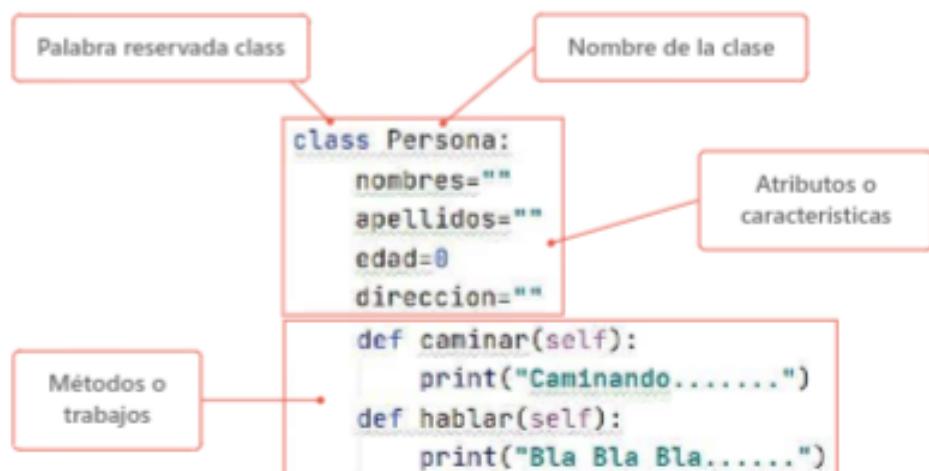
1. Primero, se crea la clase.



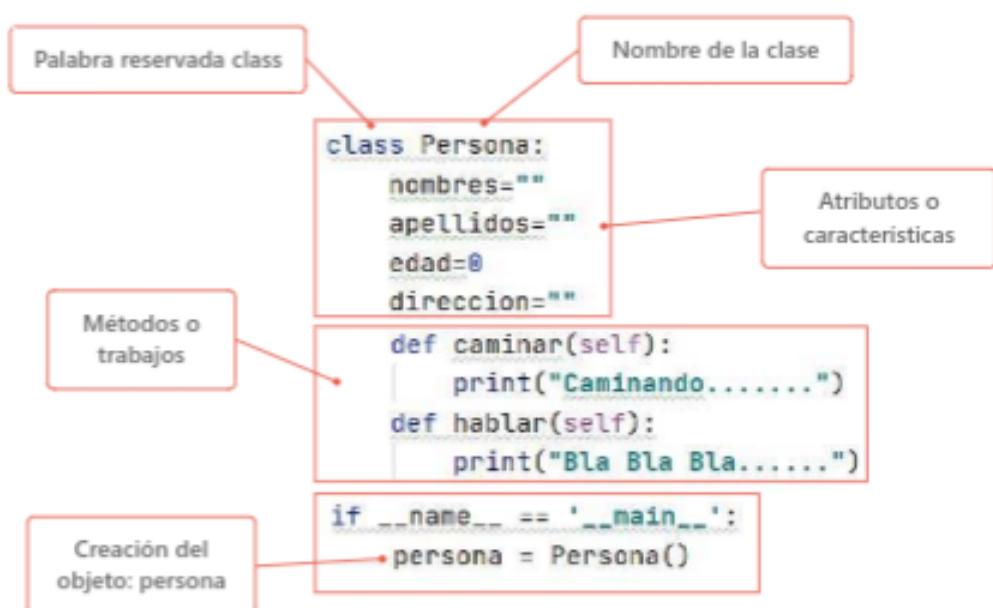
2. Luego, se indica los atributos o características de una persona.



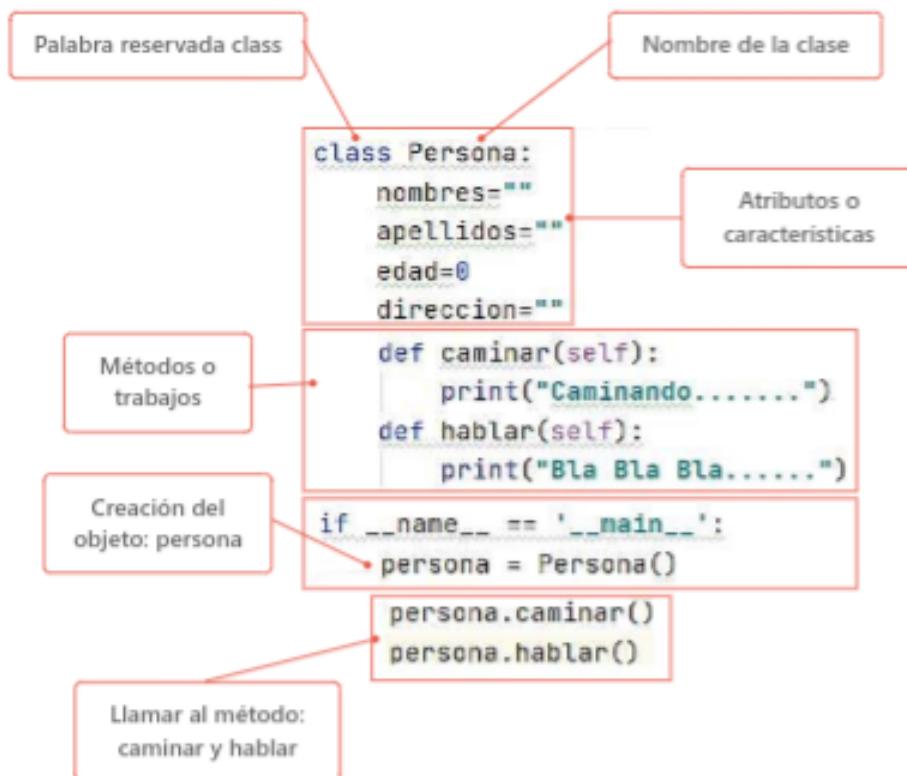
3. Se define los métodos o trabajos que realiza una persona.



4. Se crea el objeto: persona.



5. Se debe llamar al método "caminar y hablar".



6. Finalmente, se accede a sus atributos o características.



Nota

El parámetro `self` se refiere al objeto instanciado de la clase sobre la cual se está invocando dicho método.

Persona.py

```
#programa : Persona.py
#autor : jorge nolasco valenzuela
#fecha : 1-5-2020
#versión 1.0

descripcion : este programa muestra
el uso de clases - Creacion de la Clase Persona
con sus respectivos atributos y metodos

class Persona:
    nombres="" #atributo nombres
    apellidos="" #atributo apellidos
    edad=0 #atributo edad
    direccion="" #atributo direccion
    # metodo caminar
    def caminar(self):
        print("Caminando.....")
    # metodo hablar
    def hablar(self):
        print("Bla Bla Bla.....")

if __name__ == '__main__':
    #crear el objeto persona
    persona = Persona()
    #llamar al metodo caminar
    persona.caminar()
    # llamar al metodo hablar
    persona.hablar()
    # accedemos a los atributos
    persona.nombres="jorge santiago"
    persona.apellidos = "nolasco valenzuela"
    persona.edad = 48
    persona.direccion = "jr. abc"
    print(persona.nombres,persona.apellidos
          ,persona.edad,persona.direccion)
```

Caminando.....

Bla Bla Bla.....

jorge santiago nolasco valenzuela 48 jr. abc

Se observa que cada método siempre lleva como primer parámetro a `self`, y luego puede llevar más cantidad de parámetros que se desee.

En Python no existe la palabra reservada "this" como en lenguajes Java o C/C++ donde es usado para hacer referencia al propio objeto instanciado. Por su parte, Python hace uso de un parámetro

llamado `self`. Se coloca antes de cualquier otro en la definición de los métodos de la clase. Además, existen los llamados "métodos especiales". Estos son algunos ejemplos:

<code>_init_</code>	<code>_new_</code>	<code>_str_</code>	<code>_len_</code>	<code>_cmp_</code>
---------------------	--------------------	--------------------	--------------------	--------------------

Estos métodos tienen un significado especial para el intérprete de Python. Son usados en determinados casos. Por ejemplo, cuando el objeto se construye e inicializa. Asimismo, cuando se intenta representarlo como cadena, compararlo con otro objeto y transformarlo a otro tipo de dato.

El método `_init_` es uno de los métodos principales en la definición de un objeto. Este método se efectúa primero cuando se instancia un objeto, ya que sirve para inicializar los atributos y el estado del objeto. Se comporta como un constructor más no lo es, debido a que no construye el objeto, solo inicializa sus atributos. Es decir, este método es ejecutado cuando el objeto ya fue creado.

A continuación, se creará la clase Contador, la cual posee métodos de obtener incremento y reinicio de un valor.

Contador.py

```
#programa : Contador.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#####
descripcion : este programa muestra
el uso de clases - Creacion de la Clase Contador
#####

class Contador:
    #constructor de la clase
    def __init__(self):
        self.valor=0
    #metodo obtener
    def obtener(self):
        return self.valor
    # metodo incrementar
    def incrementar(self):
        self.valor=self.valor+1
    # metodo reiniciar
    def reiniciar(self):
        self.valor=0

if __name__ == '__main__':
    # crear el objeto contador
    contador = Contador()
    # llamar al metodo incrementar
    contador.incrementar()
    # llamar al metodo obtener
    var1=contador.obtener()
    #muestra el contenido de la variable var1
    print(var1)
```

4.1.6 Explicación del código contador.py

Nombre de la clase	Contador
Atributos	Valor
Métodos	<ul style="list-style-type: none"> • Obtener • Incrementar • Reiniciar

Se comienza implementando la clase Contador.

```
class Contador:
```

Definición del método obtener que retorna el contenido del atributo valor.

```
def obtener(self):
    return self.valor
```

Definición del método incrementar. Incrementa en una unidad el contenido del atributo valor.

```
def incrementar(self):
    self.valor=self.valor+1
```

```
def reiniciar(self):
    self.valor=0
```

Definición del método reiniciar. Reinicia el contenido del atributo valor en cero.

Se crea el objeto contador que viene a ser una copia de la clase Contador.

```
contador = Contador()
```

Se llama al método incrementar, el cual incrementa en una unidad el atributo valor.

```
contador.incrementar()
```

Se recupera el último contenido del atributo valor y se asigna a la variable var1.

```
var1=contador.obtener()
print(var1)
```

Mostrar el contenido de la variable var1.

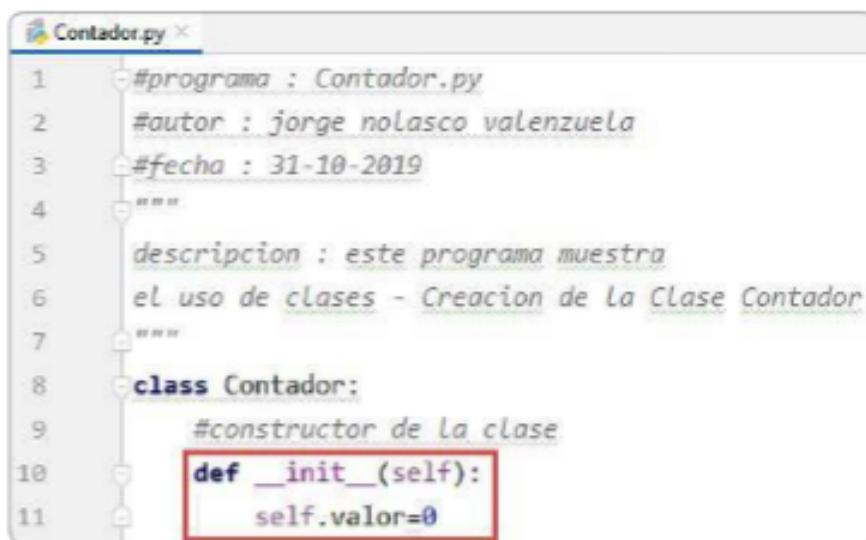
4.1.7 Definición de clases-constructor

Un constructor define e inicializa la instancia de los atributos de un objeto. El constructor se llama automáticamente cada vez que se crea un objeto.

Para crear una instancia de una clase, se utiliza el nombre de la clase como si fuera una función Junto con los argumentos requeridos por el constructor. Para crear una instancia en la clase Contador, se usa el siguiente comando:

```
contador = Contador()
```

Ahora, se indicará que el contenido del atributo valor empieza con el valor de uno, por ello, en la clase Contador, se definirá su constructor:



```

Contador.py ×
1 #programa : Contador.py
2 #autor : jorge nolasco valenzuela
3 #fecha : 31-10-2019
4 """
5     descripcion : este programa muestra
6         el uso de clases - Creacion de La Clase Contador
7 """
8 class Contador:
9     #constructor de la clase
10    def __init__(self):
11        self.valor=0

```

Se creará la clase Factura, la cual permita:

- Ingresar datos de un ítem a facturar
- Calcular el bruto, neto e impuestos
- Mostrar la factura a emitir
- Anular una factura
- Ver una factura
- Obtener bruto, impuestos y neto

Nombre de la clase	Factura
Atributos	Número Fecha Cliente Impuesto IdArticulo Descripción Precio Cantidad

	Bruto Montolimp Neto Estado
Métodos	<ul style="list-style-type: none"> • Ingresar • Calcular • Anular • Ver • ObtenerB • ObtenerI • ObtenerN • Anular

Factura.py

```
#programa : Factura.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de clases
Creacion de la Clase Factura
"""

class Factura:
    #constructor de la clase
    def __init__(self):
        self.numero="""
        self.fecha="""
        self.cliente="""
        self.impuesto=0
        self.articulo="""
        self.precio=0
        self.cantidad=0
        self.bruto=0
        self.montolimp = 0
        self.neto = 0
        self.estado = True
    #metodo ingresar
    def ingresar(self):
        print("INGRESE FACTURA")
        self.numero=input("# Factura:")
        self.fecha=input("Fecha:")
        self.cliente=input("Cliente:")

        self.impuesto=int(input("Impuesto:"))
        self.articulo=input("Articulo:")
        self.precio=int(input("Precio:"))
        self.cantidad=int(input("Cantidad:"))
        print("FIN INGRESO")
```

```
#metodo calcular
def calcular(self):
    self.bruto = self.precio *\n        self.cantidad
    self.montoImp = self.bruto *\n        (self.impuesto/100)
    self.neto = self.bruto +\n        self.montoImp
print("FACTURA CALCULADA")
    print("Bruto:{0:.2f}"
        .format(self.bruto))
    print("Impuesto:{0:.2f}"
        .format(self.montoImp))
    print("Neto:{0:.2f}"
        .format(self.neto))
#metodos obtener
def obtenerB(self):
    return self.bruto
def obtenerI(self):
    return self.montoImp
def obtenerN(self):
    return self.neto
def anular(self):
    self.estado=False
    print("FACTURA ANULADA")

if __name__ == '__main__':
    factura = Factura()
    while True:
        #menu de opciones
        print("MENU:")
        print("1: INGRESAR")
        print("2: CALCULAR")
        print("3: ANULAR")
        print("4: SALIR")
        opc=int(input
            ("SELECCIONE OPCION:"))
        if opc==1:
            #ingresar factura
            factura.ingresar()
        elif opc==2:
            #calcular factura
            factura.calcular()
        elif opc==3:
            # anular factura
            factura.anular()
        else:
            break
```

MENU:**1: INGRESAR****2: CALCULAR****3: ANULAR****4: SALIR****SELECCIONE OPCION :1****INGRESE FACTURA**

Factura:0001

Fecha:11/05/2017

Cliente: GLORIA

Impuesto:18

Articulo: HOJAS

Precio:10

Cantidad:10

FIN INGRESO**MENU:****1: INGRESAR****2: CALCULAR****3: ANULAR****4: SALIR****SELECCIONE OPCION :2****FACTURA CALCULADA**

Bruto:100.00

Impuesto:18.00

Neto:118.00

MENU:**1: INGRESAR****2: CALCULAR****3: ANULAR****4: SALIR****SELECCIONE OPCION :4**

4.1.8 Ejemplos

Se creará la clase Profesor con sus principales atributos y algunos métodos:

Profesor.py

```
#programa : Profesor.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de clases - Creacion de la
Clase Profesor
"""

class Profesor:
    #definicion de atributos
    nombres= ""
    apellidos= ""
```

```

edad=0
direccion=""
especialidad=""
#constructor de la clase
def __init__(self):
    self.nombres="juan"
    self.apellidos="navarro"
    self.edad=33
    self.direccion="av xyz"
    self.especialidad="sistemas"
#metodo darLecciones
def darLecciones(self):
    print("Alumnos Bla Bla")
# metodo datos
def datos(self):
    print("Nombres :" +self.nombres)
    print("Apellidos :" +self.apellidos)
    print("Edad :" +str(self.edad))
    print("Direccion :" +self.direccion)
    print("Especialidad :" +self.especialidad)

if __name__ == '__main__':
    #creamos el objetos profesor
    profesor = Profesor()
    #llamamos al metodo datos
    profesor.datos()

```

Nombres :juan
Apellidos :navarro
Edad :33
Direccion :av xyz
Especialidad :sistemas

Se creará la clase Teléfono con sus principales atributos y algunos métodos:

Telefono.py
<pre> #programa : Telefono.py #autor : jorge nolasco valenzuela #fecha : 01-05-2020 #MM descripcion : este programa muestra el uso de clases - Creacion de la Clase Telefono #MM class Telefono: #definicion de atributos marca="" modelo="" numero=0 </pre>

```
#constructor de la clase
def __init__(self):
    self.marca="samsung"
    self.modelo="S8"
    self.numero=999999999
#metodo llamar
def llamar(self):
    print("LLamando ....")
# metodo colgar
def colgar(self):
    print("Colgar.....")
# metodo datos
def datos(self):
    print("Marcar :" +self.marca)
    print("Modelo :" + self.modelo)
    print("Numero :" + str(self.numero))

if __name__ == '__main__':
    #creamos el objetos telefono
    telefono = Telefono()
    #llamamos al metodo datos
    telefono.datos()
```

**Marcar :samsung
Modelo :S8
Numero :999999999**

Se creará la clase Libro con sus principales atributos y algunos métodos:

Libro.py
<pre>#programa : Libro.py #autor : jorge nolasco valenzuela #fecha : 01-05-2020 #versión 1.0 descripcion : este programa muestra el uso de clases Creacion de la Clase Libro class Libro: #definicion de atributos isbn="" titulo="" editorial="" año=0 precio=0 #constructor de la clase def __init__(self): #atributos self.isbn="978-612-46976-0-9" self.titulo="Python-Guia Completa" self.editorial = "INKADROID"</pre>

```

self.ano = 2017
self.precio = 50
#metodo prestar libro
def prestar(self):
    print("Prestar Libro ....")
#metodo devolver libro
def devolver(self):
    print("Devolver Libro ...")
#metodo datos
def datos(self):
    print("ISBN:" +self.isbn)
    print("TITULO:" +self.titulo)
    print("EDITORIAL:" +self.editorial)
    print("AÑO:" +str(self.ano))
    print("PRECIO:" +str(self.precio))

if __name__ == '__main__':
    #Creamos el objetos libro
    libro = Libro()
    #llamamos al metodo datos
    libro.datos()

```

ISBN :978-612-46976-0-9
TITULO :Python - Guia Completa
EDITORIAL :INKADROID
AÑO :2017
PRECIO :50

4.1.9 Atributo __dict__

Cada objeto en Python tiene un atributo denotado por `__dict__`. Este diccionario/objeto tipo diccionario contiene todos los atributos definidos para el objeto mismo. Asimismo, asigna el nombre del atributo a su valor.

listarAtributos.py

```

#programa : listarAtributos.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripcion : este programa muestra
el uso de clases - Creacion de la Clase Persona
y listar respectivos atributos y metodos
"""

class Persona:
    nombres="Juan" #atributo nombres
    apellidos="Perez Perez" #atributo apellidos
    edad=30 #atributo edad
    direccion="Jr. abc" #atributo direccion
    #metodo caminar
    def caminar(self):
        print("Caminando.....")

```

```
#metodo hablar
def hablar(self):
    print("Bla Bla Bla.....")
if __name__ == '__main__':
    #crear el objeto persona
    persona=Persona()
    #llamar al metodo caminar
    print(Persona.__dict__)

(__module__: '_main_', 'nombres': 'Juan', 'apellidos': 'Perez Perez', 'edad': 30, 'direccion': 'Jr.
abc', 'caminar': <function Persona.caminar at 0x0114BC48>, 'hablar': <function Persona.hablar
at 0x0114BC90>, '__dict__': <attribute '__dict__' of 'Persona' objects>, '__weakref__': <attribute '__
weakref__' of 'Persona' objects>, '__doc__': None}
```

4.1.10 Herencia simple

La herencia es una característica que permite la creación de clases a partir de otras. Esto conlleva la reutilización del código y la especialización de las clases. La reutilización del código permite que se puede definir clases nuevas partiendo de otras ya existentes. En este caso, se heredan sus propiedades y métodos. A continuación, se muestra un ejemplo de herencia:

Clase padre: Trabajador

Clase hija: Empleado



Herencia1.py

```
#programa : Herencia1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#descripción : este programa muestra
#el uso de clases - Herencia
#clase padre-Trabajador
class Trabajador:
    def __init__(self,nom,dir,tel,doc):
        self.nom=nom
        self.dir=dir
        self.tel=tel
        self.doc=doc
```

```

#metodo datos
def datos(self):
    print("Nombres : ",self.nom)
    print("Direccion : ",self.dir)
    print("Telefono : ",self.tel)
    print("Documento : ",self.doc)

#clase hija - Empleado
class Empleado(Trabajador):
    def __init__(self,nom,dir,tel,doc,suel,horast):
        Trabajador.__init__(self,nom,dir,tel,doc)
        self.suel=suel
        self.horast=horast

#metodo datos
def datos(self):
    print("Nombres del Empleado: ",self.nom)
    print("Direccion del Empleado: ",self.dir)
    print("Telefono del Empleado: ",self.tel)
    print("Documento del Empleado: ",self.doc)

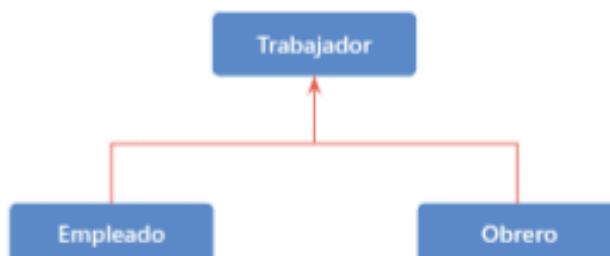
#metodo calcularSu
def calcularSu(self):
    pagoxhora=self.suel/30/8
    return pagoxhora * self.horast

if __name__ == '__main__':
    #Creamos el objeto empleado
    empleado = Empleado("luis","av xyz"
                         ,"2655011","19168210",1000,240)
    #Llamamos al metodo datos
    empleado.datos()
    #Llamamos al metodo calcularSu
    sueldo=empleado.calcularSu()
    #Mostramos el sueldo
    print("sueldo:{0:5.0f}"
          .format(sueldo))

```

Nombres del Empleado: luis
 Direccion del Empleado: av xyz
 Telefono del Empleado: 2655011
 Documento del Empleado: 19168210
 Sueldo: 1000

Ahora, se añadirá la clase hija Obrero:



Herencia2.py

```
#programa : Herencia2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#MM

descripcion : este programa muestra
el uso de Herencia

#clase padre - Trabajador
class Trabajador:
    #constructor de la clase
    def __init__(self,nom,dir,tel,doc):
        self.nom=nom
        self.dir=dir
        self.tel=tel
        self.doc=doc
    #metodo datos
    def datos(self):
        print("Nombres : ",self.nom)
        print("Direccion : ",self.dir)
        print("Telefono : ",self.tel)
        print("Documento : ",self.doc)

#clase hija - Empleado
class Empleado(Trabajador):
    # constructor de la clase
    def __init__(self,nom,dir,tel,
                 doc,suel,horast):
        Trabajador.__init__(self,nom,dir,tel,doc)
        self.suel=suel
        self.horast=horast
    #metodo datos
    def datos(self):
        print("Nombres Empleado:",self.nom)
        print("Direccion Empleado:",self.dir)
        print("Telefono Empleado:",self.tel)
        print("Documento Empleado:",self.doc)

    #metodo calcularSu
    def calcularSu(self):
        pagoxhora=self.suel/30/8
        return pagoxhora * self.horast

#clase hija - Obrero
class Obrero(Trabajador):
    # constructor de la clase
    def __init__(self,nom,dir,tel,
                 doc,sal,horast):
        Trabajador.__init__(self,nom,dir,tel,doc)
        self.sal=sal
        self.horast=horast
    #metodo datos
    def datos(self):
        print("Nombres del Obrero:",self.nom)
```

```

print("Direccion del Obrero:",self.dir)
print("Telefono del Obrero:",self.tel)
print("Documento del Obrero:",self.doc)
#metodo calcularSa
def calcularSa(self):
    pagoxhora=self.sal/7/8
    return pagoxhora * self.horast
if __name__ == '__main__':
    #Creamos el objeto empleado
    empleado = Empleado("luis","av xyz","2655011","19168210",1000,240)
    #Llamamos al metodo datos
    empleado.datos()
    #Llamamos al metodo calcularSu
    sueldo=empleado.calcularSu()
    #Imprimimos el sueldo
    print("Sueldo:{0:5.0f}".format(sueldo))
    print("====")
    #Objeto obrero
    obrero = Obrero("luis","av xyz","2655011","19168210",250,48)
    #Llamamos al metodo datos
    obrero.datos()
    #Llamamos al metodo calcularSa
    salario=obrero.calcularSa()
    #Imprimimos el salario
    print("Salario:{0:5.0f}"
          .format(salario))

```

Nombres Empleado: luis
 Dirección Empleado: av xyz
 Telefono Empleado: 2655011
 Documento Empleado: 19168210
 Sueldo: 1000
 =====
 Nombres Obrero: luis
 Dirección Obrero: av xyz
 Telefono Obrero: 2655011
 Documento Obrero: 19168210
 Salario: 214

Luego, se implementa los métodos get y set:

Herencia3.py
<pre> #programa : Herencia3.py #autor : jorge nolasco valenzuela #fecha : 01-05-2020 """ descripcion : este programa muestra el uso de clases - Herencia """ #clase padre Trabajador class Trabajador: </pre>

```
def __init__(self):
    self.nom = ""
    self.dir = ""
    self.tel = ""
    self.doc = ""
def getNom(self):
    return self.nom
def setNom(self,nuevoValor):
    self.nom=nuevoValor
def getDir(self):
    return self.dir
def setDir(self,nuevoValor):
    self.dir=nuevoValor
def getTel(self):
    return self.tel
def setTel(self,nuevoValor):
    self.tel=nuevoValor
def getDoc(self):
    return self.doc
def setDoc(self,nuevoValor):
    self.doc=nuevoValor
def datos(self):
    print("Nombres : ",self.nom)
    print("Direccion : ",self.dir)
    print("Telefono : ",self.tel)
    print("Documento : ",self.doc)
#clase hija Empleado
class Empleado(Trabajador):
    def __init__(self):
        Trabajador.__init__(self)
        self.suel=0
        self.horast=0
    def getSuel(self):
        return self.suel
    def setSuel(self,nuevoValor):
        self.suel=nuevoValor
    def getHorast(self):
        return self.horast
    def setHorast(self,nuevoValor):
        self.horast=nuevoValor
    def datos(self):
        print("Nombres Empleado: ",self.nom)
        print("Direccion Empleado: ",self.dir)
        print("Telefono Empleado: ",self.tel)
        print("Documento Empleado: ",self.doc)
    def calcularSu(self):
        pagoxhora=self.suel/30/8
        return pagoxhora * self.horast
#clase hija Obrero
class Obrero(Trabajador):
```

```
def __init__(self):
    Trabajador.__init__(self)
    self.sal=0
    self.horast=0
def getSal(self):
    return self.sal
def setSal(self,nuevoValor):
    self.sal=nuevoValor
def getHorast(self):
    return self.horast
def setHorast(self,nuevoValor):
    self.horast=nuevoValor
def datos(self):
    print("Nombres Obrero:",self.nom)
    print("Direccion Obrero:",self.dir)
    print("Telefono Obrero:",self.tel)
    print("Documento Obrero:",self.doc)
def calcularSa(self):
    pagoxhora=self.sal/7/8
    return pagoxhora * self.horast
if __name__ == '__main__':
    nombre=input("Ing Nombre Empleado :")
    direccion=input("Ing Direccion Empleado :")
    telefono=input("Ing Telefono Empleado :")
    documento=input("Ing Documento Empleado :")
    sueldo=int(input("Ing Sueldo Empleado :"))
    horasMensual=int(input("Ing Horas Mensuales"
                           " Trabajadas de Empleado :"))

    #Creamos objeto empleado
    empleado = Empleado()
    empleado.setNom(nombre)
    empleado.setDir(direccion)
    empleado.setTel(telefono)
    empleado.setDoc(documento)
    empleado.setSuel(sueldo)
    empleado.setHorast(horasMensual)
    empleado.datos()
    sueldo=empleado.calcularSu()
    print("Sueldo:{0:5.0f}".format(sueldo))

print("=====")
    nombre=input("Ing Nombre Obrero :")
    direccion=input("Ing Direccion Obrero:")
    telefono=input("Ing Telefono Obrero :")
    documento=input("Ing Documento Obrero :")
    salario=int(input("Ing Salario Semanal Obrero :"))
    horaSemanal=int(input("Ing Horas Semanales"
                          " Trabajadas Obrero :"))

    #Creamos objeto obrero
    obrero = Obrero()
    obrero.setNom(nombre)
    obrero.setDir(direccion)
```

```
obrero.setTel(telefono)
obrero.setDoc(documento)
obrero.setSal(salario)
obrero.setHorast(horaSemanal)
obrero.datos()
salario=obrero.calcularSa()
print("Salario:{0:5.0f}"
      .format(salario))
```

```
Ingrese Nombre Empleado :Juan Perez Perez
Ingrese Direccion Empleado :Jr. abc
Ingrese Telefono Empleado :2645010
Ingrese Documento Empleado :09668210
Ingrese Sueldo Empleado :1000
Ingrese Horas Mensuales Trabajadas Empleado :240
Nombres Empleado: Juan Perez Perez
Direccion Empleado: Jr. abc
Telefono Empleado: 2645010
Documento del Empleado: 09668210
Sueldo: 1000
=====
Ingrese Nombre Obrero: Jaime Lara Lara
Ingresar Direccion Obrero:av. xyz
Ingresar Telefono Obrero :2645011
Ingresar Documento Obrero :09668220
Ingresar Salario Semanal Obrero :250
Ingresar Horas Semanales Trabajadas Obrero :60
Nombres Obrero: Jaime Lara Lara
Direccion Obrero: av. xyz
Telefono Obrero: 2645011
Documento Obrero: 09668220
Salario: 268
```

4.1.11 Herencia múltiple

En Python, la herencia múltiple se define como una lista de superclases. Es ordenada como una especie de algoritmo por orden de profundidad y ejecuta el primer método que encuentra. A continuación, se muestra un ejemplo de herencia múltiple:



Herencia4.py

```
#programa : Herencia4.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#####
descripcion : este programa muestra
el uso de clases - Herencia Multiple
#####

class Camara:
    def sacar_Foto(self):
        print("Fotografiando")
class Reproductor:
    def reproducir(self):
        print("Reproducir MP3")
class Despertador:
    def despertar(self):
        print("trrrriiii!!!!")
class Grabadora:
    def grabando(self):
        print("Bla Bla Bla Bla!!!!")
class Celular(Camara,Reproductor
    ,Despertador,Grabadora):
    pass
if __name__ == '__main__':
    #crear objeto celular
    celular=Celular()
    #llamar al metodo sacar_Foto
    celular.sacar_Foto()
    # llamar al metodo reproducir
    celular.reproducir()
    # llamar al metodo despertar
    celular.despertar()
    # llamar al metodo grabando
    celular.grabando()
```

Fotografiando**Reproducir MP3****trrrriiii!!!!****Bla Bla Bla Bla!!!!****4.1.12 Conocer un objeto de una clase hija**

La función `issubclass()` verifica si una clase específica es hija de otra. Ejemplo:

Herencia5.py

```
#programa : Herencia5.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#####
descripcion : este programa muestra
el uso de la funcion issubclass
#####
```

```
#clase padre-Trabajador
class Trabajador:
    pass
#clase hija - Empleado
class Empleado(Trabajador):
    pass

if __name__ == '__main__':
    print(isinstance(Empleado,Trabajador))
```

True

4.1.13 Conocer un objeto de una clase específica

La función `isinstance()` verifica si un objeto es de una clase específica. Ejemplo:

Herencia6.py

```
#programa : Herencia6.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""

descripcion : este programa muestra
el uso de la funcion isinstance
"""

#clase padre-Trabajador
class Trabajador:
    pass
#clase hija - Empleado
class Empleado(Trabajador):
    pass

if __name__ == '__main__':
    empleado=Empleado()
    print(isinstance(empleado,Empleado))
```

True

4.1.14 Otros ejemplos de uso de herencia

A continuación, se muestran otros ejemplos:

Herencia7.py

```
# programa : Herencia7.py
# autor : jorge nolasco valenzuela
# fecha : 01-05-2020
"""

descripcion : este programa muestra
el uso de herencia
"""

# clase padre
class Padre:
```

```

def __init__(self, nombre):
    self.nombre = nombre
def datos(self):
    print("Mi Nombre es : ", self.nombre)
# clase hija
class Hija(Padre):
    def __init__(self, nombre):
        Padre.__init__(self, nombre)

```

```

hija = Hija("Jorge")
hija.datos()

```

Mi Nombre es : Jorge

Herencia8.py

```

# programa : Herencia8.py
# autor : jorge nolasco valenzuela
# fecha : 01-05-2020
"""
descripción : este programa muestra
el uso de herencia
usando el metodo __str__
"""

# clase padre
class Padre:
    def __init__(self, nombre):
        self.nombre = nombre

    def __str__(self):
        print("Mi Nombre es : ", self.nombre)

# clase hija
class Hija(Padre):
    def __init__(self, nombre):
        Padre.__init__(self, nombre)

```

```

hija = Hija("Jorge")

```

Mi Nombre es : Jorge

4.1.15 Iteradores y generadores

Los "iteradores" y "generadores" son objetos que cuentan con el método `_next_()`, el cual regresa una serie de objetos de uno en uno cada vez que es invocado.

A continuación, se muestran algunos ejemplos de iteradores.

Se muestran los elementos de una lista de la siguiente forma clásica:

```

lista = [1,2,3,4,5,6,7,8,9,10]
print(lista)

```

Luego con iteradores:

```
lista = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10].__iter__()
while True:
    try:
        elemento = lista.__next__()
        print(elemento,end="")
    except StopIteration:
        break
```

Los generadores son una forma sencilla y potente de iterador. Un generador es una función especial que produce secuencias completas de resultados en lugar de ofrecer un único valor.

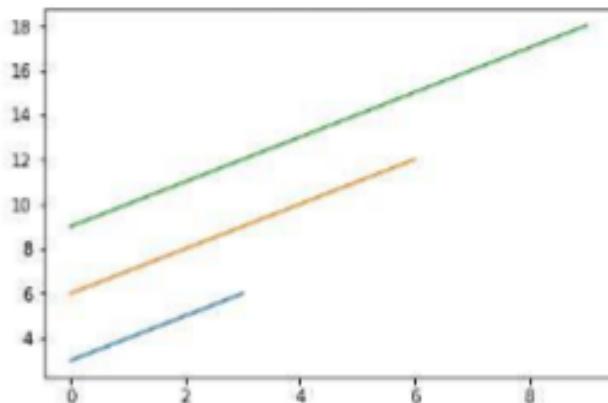
A continuación, se muestran algunos ejemplos de generador:

```
def funcion(n):
    for i in range(n):
        yield i

for elemento in funcion(10):
    print(elemento,end="")
```

0123456789

4.1.16 Ejemplo de métodos especiales



Como se dijo anteriormente, estos métodos tienen un significado especial para el intérprete de Python, pues, son usados en determinados casos. A continuación, se muestra un ejemplo:

MetodosEspeciales.py

```
#programa : MetodosEspeciales.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#www

descripcion : este programa muestra
el uso de Clases metodos especiales
emulando operaciones matematicas
sobre puntos
#www
```

```

from math import hypot
class Spunto:
    #metodo encargado de las tareas
    #de inicializacion
    def __init__(self,x=0,y=0):
        self.x=x
        self.y=y
    #retorna un string que describe el
    #objeto segun un formato por defecto
    def __repr__(self):
        return (self.x,self.y)
    def __abs__(self):
        return hypot(self.x,self.y)
    def __bool__(self):
        return bool(abs(self))
    #devuelve la suma de dos puntos
    def __add__(self, other):
        return Spunto(self.x + other.x,
                      self.y + other.y)
    #Devuelve la resta de ambos puntos
    def __sub__(self, other):
        return Spunto(self.x - other.x,
                      self.y - other.y)
    #Devuelve la multiplicacion de
    #ambos puntos
    def __mul__(self,scalar):
        return Spunto\
            (self.x*scalar,self.y*scalar)
    #Muestra el punto como un par ordenado
    def __str__(self):
        return "(" + str(self.x) + \
               ", " + str(self.y) + ")"
if __name__ == '__main__':
    #creacion de objetos puntos
    punto1=Spunto(3,6)
    punto2=Spunto(6,12)
    print(punto1 + punto2)

```

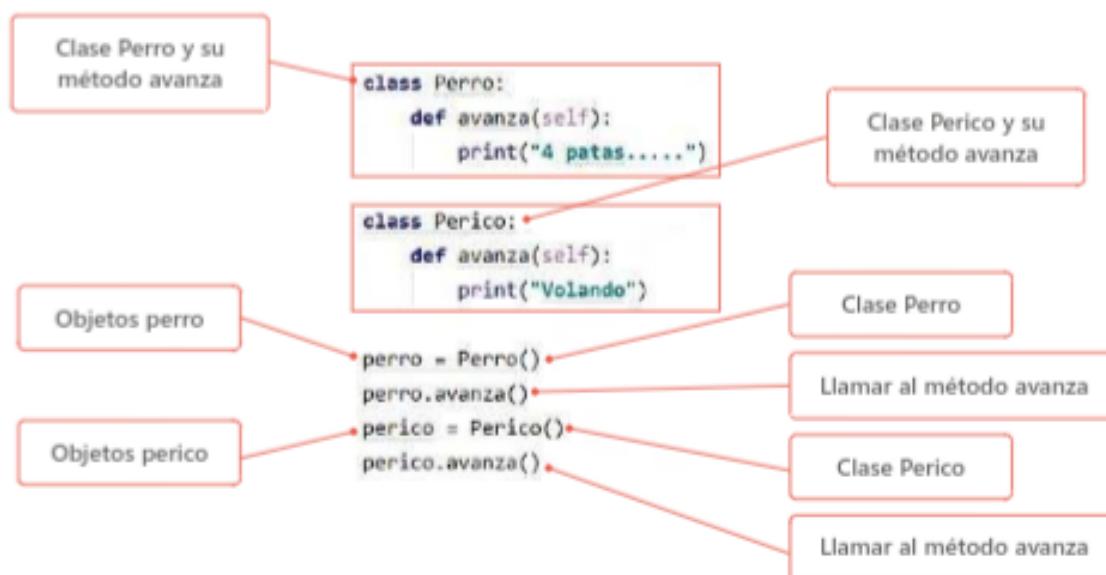
(9, 18)

4.1.17 Polimorfismo

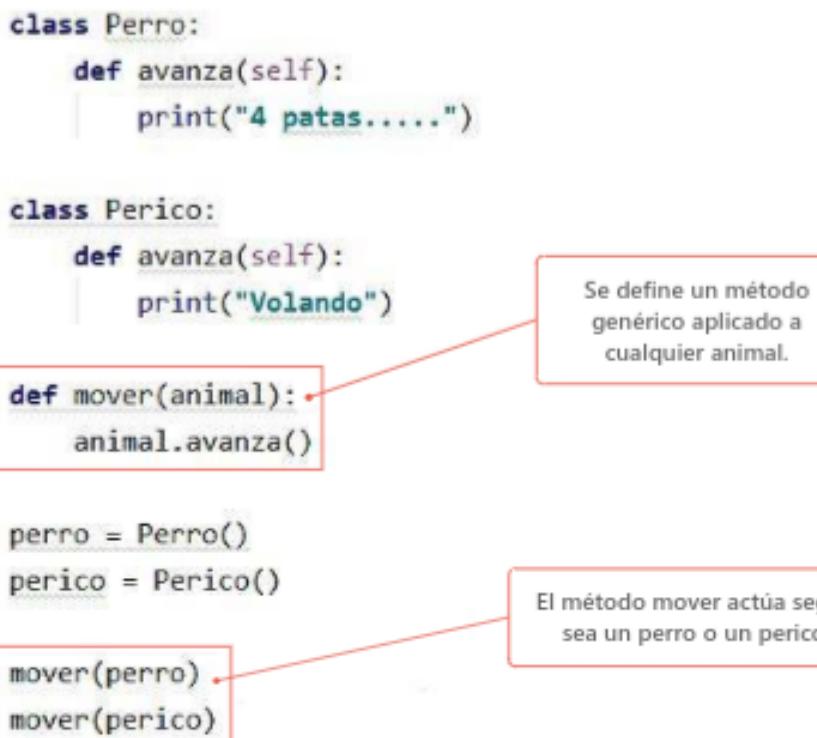
En el ámbito de la orientación a objetos, el polimorfismo hace referencia a la habilidad que tienen los objetos de diferentes clases a responder a métodos con el mismo nombre, pero con implementaciones diferentes.

A continuación, se definen dos clases que poseen métodos con el mismo nombre, pero con diferente funcionalidad:

» Sin polimorfismo



» Con polimorfismo



5

Manejo de ficheros

5.1 Manejo de ficheros

Uno de los problemas más comunes es procesar los datos almacenados en archivos, porque están almacenados en discos duros, usbs, etc.

Si se quiere implementar una base de datos simple, la única forma de almacenar la información entre ejecuciones del programa es guardándola en un archivo.

Ahora, se observará cómo manejar archivos desde los programas. Existen dos formas básicas de acceder a un archivo: una es utilizarlo como un archivo de texto, que se procesará línea por línea; la otra es tratarlo como un archivo binario, que se procesará byte por byte. En Python, se utiliza la función open para abrir un archivo:

```
fichero = open("hola.txt","r",encoding=None)
```

Ahora, se analizará la línea anterior:

La función (open) apertura el archivo. Si la apertura es exitosa, la función devuelve un objeto; de lo contrario, se genera una excepción (por ejemplo, FileNotFoundError si el archivo que va a leer no existe).

El segundo parámetro (mode) especifica el modo abierto, en este caso read (lectura).

```
fichero = open("hola.txt","r",encoding=None)
```

Este parámetro especifica el nombre del archivo a abrir.

El tercer parámetro (encoding) especifica el tipo de codificación (por ejemplo, UTF-8 cuando se trabaja con archivos de texto).

Nota

El modo y los argumentos pueden omitirse, pues, sus valores predeterminados se suponen entonces. El modo de apertura predeterminado es leer en modo de texto.

La operación más sencilla a realizar sobre un archivo es leer su contenido:

```
for linea in fichero:  
    print(linea)
```

Para obtener todas las líneas del archivo, se puede utilizar la función:

```
lineas = fichero.readlines()
```

Ahora, las operaciones básicas de un archivo:

Apertura el archivo.txt para lectura

```
fichero = open("fichero.txt","r")
```

Apertura el archivo.txt para escritura

```
fichero = open("fichero.txt","w")
```

Apertura el archivo.txt para añadir

```
fichero = open("archivo.txt", "a")
```

Apertura el archivo.txt para lectura y actualización

```
fichero = open("archivo.txt", "r+")
```

Apertura el archivo.txt para escribir y actualización

```
fichero = open("archivo.txt", "w+")
```

Borra el contenido del archivo: archivo.txt

```
fichero.truncate()
```

Escribiendo contenido en el archivo

```
fichero.write("escribiendo esta linea")
```

Cerrar el archivo

```
fichero.close ()
```

5.1.1 Ejercicios de archivos TXT

5.1.1.1 Lectura de archivos

Fichero1.py

```
#programa : Fichero1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
##
descripcion : este programa muestra
la lectura de un fichero
##
#apertura del fichero a leer
fichero = open("hola.txt","r")
#leer fichero
for linea in fichero:
    print(linea)
#cerrar fichero
fichero.close()
```

**Aprendiendo a
Programar
con Python 3
Autor:Jorge Nolasco Valenzuela**

Ahora, el anterior ejemplo con un manejador de excepciones:

```
#programa : Fichero1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
la lectura de un fichero - con excepciones
"""

try:
    #apertura del fichero a leer
    fichero = open("hola.txt","r")
    #leer fichero
    for linea in fichero:
        print(linea)
    #cerrar fichero
    fichero.close()
except Exception as e:
    print("no se puede aperturar el archivo",e)
```

**Aprendiendo a
Programar
con Python 3
Autor:Jorge Nolasco Valenzuela**

5.1.1.2 Lectura utilizando readline()

```
#programa : Fichero1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
la lectura de un fichero utilizando readline
"""

try:
    ccnt = lcnt = 0
    fichero = open('hola.txt', 'rt')
    linea = fichero.readline()
    while linea != "":
        for ele in linea:
            print(ele, end="")
        linea = fichero.readline()
    fichero.close()
except IOError as e:
    print("no se puede aperturar el archivo:", e)
```

Bienvenido
Aprendiendo
Python
Autor:Jorge Nolasco Valenzuela

5.1.1.3 Escritura de archivos

```
#programa : Fichero2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#versión 1.0

descripcion : este programa muestra
escritura de fichero

#apertura del fichero a escribir
texto = open("hola.txt","w")
#escribir fichero
texto.write("Bienvenido\n")
texto.write("Aprendiendo\n")
texto.write("Python\n")
texto.write("Autor:Jorge Nolasco Valenzuela\n")
#cerrar fichero
texto.close()
```

A continuación, otro ejemplo de escritura de archivo:

```
#programa : Fichero3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#versión 1.0

descripcion : este programa muestra
escritura de fichero

try:
    fichero = open('numeros.txt', 'w')
    for i in range(10):
        numero = str(i+1)
        for ele in numero:
            fichero.write(ele)
            print(numero,end="-")
    fichero.close()
except Exception as e:
    print("no se puede aperturar el archivo:", e)
```

1-2-3-4-5-6-7-8-9-10-

5.1.2 Ejercicios de archivos binarios

Existen una serie de datos amorfos, pues son datos que no tienen una forma específica. Además, son solo una serie de bytes. Esto no significa que estos bytes no puedan tener su propio significado. Por ejemplo, los gráficos de mapa de bits si lo tienen.

Los datos amorfos no pueden almacenarse utilizando ninguno de los medios presentados anteriormente: no son cadenas ni listas. Debe haber un contenedor especial capaz de manejar dichos datos.

Python tiene más de un contenedor de este tipo. Uno de ellos tiene un nombre de clase especializado: `bytearray`. Como su nombre lo indica, es una matriz que contiene bytes (amorfos).

Ahora, se mostrará cómo escribir una matriz de bytes en un archivo binario:

Modo binario	Descripción
<code>rb</code>	leer
<code>wb</code>	escribir
<code>ab</code>	adjuntar
<code>r+b</code>	leer y actualizar
<code>w+b</code>	escribir y actualizar

5.1.2.1 Escritura de archivos

```
binario1.py
#programa : binario1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
escritura de fichero binario
"""
datos = bytearray(10)

for i in range(len(datos)):
    datos[i] = 10 + i
try:
    fichero = open('file1.bin', 'wb')
    fichero.write(datos)
    fichero.close()
except Exception as e:
    print("no se puede aperturar el archivo",e)
```

Ahora, se explicará el código anterior:

- Primero, se inicia bytearray con valores igual o mayores a 10.

Si se desea que el contenido del archivo sea claramente legible, se debe reemplazar 10, por ejemplo, con ord('a').

datos = bytearray(10)

- Luego, se crea el archivo usando open(). La única diferencia en comparación es el modo, es decir, la escritura.

fichero = open('file1.bin', 'wb')

- El método write() permite escribir los datos al fichero.

fichero.write(datos)

- Finalmente, se debe cerrar el fichero.

fichero.close()

5.1.2.2 Lectura de archivos con readinto()

La lectura de un archivo binario requiere el uso del método readinto().

```
#programa : binario2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
lectura de fichero binario
"""

datos = bytearray(10)

try:
    fichero = open('file1.bin', 'rb')
    fichero.readinto(datos)
    fichero.close()

    for ele in datos:
        print(hex(ele), end=' ')
except Exception as e:
    print("no se puede aperturar el archivo", e)

Oxa 0xb 0xc 0xd 0xe 0xf 0x10 0x11 0x12 0x13
```

Ahora, se explicará el código anterior:

- Primero, se abre el archivo (el que se creó usando el código anterior) con el modo descrito como rb.

fichero = open('file1.bin', 'rb')

- Luego, se debe leer su contenido en la matriz de bytes (denominada "data") que es de tamaño diez bytes.

```
fichero.readinto(datos)
```

- Finalmente, se imprime el contenido de la matriz de bytes.

```
for ele in datos:  
    print(hex(ele), end=' ')
```

5.1.2.3 Lectura de archivos con read()

La lectura de un archivo binario opcionalmente se puede realizar con el método `read()`.

```
#programa : binario3.py  
#autor : jorge nolasco valenzuela  
#fecha : 01-05-2020  
  
descripción : este programa muestra  
lectura de fichero binario usando read  
  
try:  
    fichero = open('file1.bin', 'rb')  
    datos = bytearray(fichero.read())  
    fichero.close()  
  
    for ele in datos:  
        print(hex(ele), end=' ')  
  
except Exception as e:  
    print("no se puede aperturar el archivo", e)  
  
0xa 0xb 0xc 0xd 0xe 0xf 0x10 0x11 0x12 0x13
```

5.1.2.4 Copiar archivos

A continuación, se debe realizar un ejemplo de copiar desde un archivo a otro.

```
binario4.py  
#programa : binario4.py  
#autor : jorge nolasco valenzuela  
#fecha : 01-05-2020  
  
descripción : este programa muestra  
copiar datos desde un archivo a otro  
  
origen = input("Ingrese Nombre del Fichero Origen: ")  
try:  
    fichero_fuente = open(origen, 'rb')  
except Exception as e:  
    print("no se puede aperturar el archivo origen: ", origen(e.errno))  
    exit(e.errno)
```

```

destino = input("Ingrese Nombre del Fichero Destino: ")
try:
    fichero_destino = open(destino, 'wb')
except Exception as e:
    print("no se puede aperturar el archivo destino: ", origen(e.errno))
    fichero_fuente.close()
    exit(e.errno)

buffer = bytearray(65536)
total = 0
try:
    readin = fichero_fuente.readinto(buffer)
    while readin > 0:
        written = fichero_destino.write(buffer[:readin])
        total += written
        readin = fichero_fuente.readinto(buffer)
except Exception as e:
    print("no se puede aperturar el archivo destino: ",e)
    exit(e.errno)

fichero_fuente.close()
fichero_destino.close()

```

Ingrese Nombre del Fichero Origen: file1.bin

Ingrese Nombre del Fichero Destino: file2.bin

5.1.3 Archivos JSON

JavaScript Object Notation (JSON) es un formato especial para el intercambio de datos. Es un formato estándar que puede ser manipulado por una amplia gama de sistemas. Por lo tanto, puede ser muy útil para transmitir datos.



A continuación, estos son algunos ejemplos de manejo de archivos JSON:

```

alimentos.json ✘
1  "Alimentos": [
2      "Fruta": [
3          {"Nombre": "Manzana", "Cantidad": 11},
4          {"Nombre": "Pera", "Cantidad": 22},
5          {"Nombre": "Naranja", "Cantidad": 33}
6      ],
7      "Verduras": [
8          {"Nombre": "Lechuga", "Cantidad": 55},
9          {"Nombre": "Tomate", "Cantidad": 55},
10         {"Nombre": "Pepino", "Cantidad": 66}
11     ]
12 ]
13
14
15
16
17
18

```

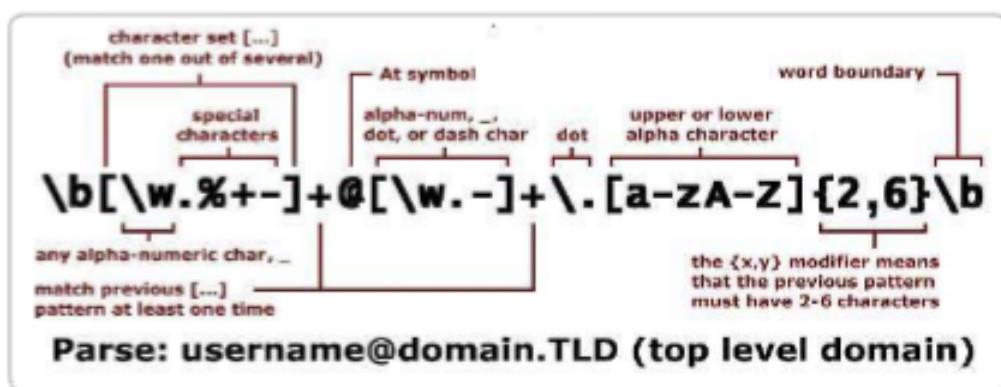
```
#programa : Json1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
la lectura de un fichero json
"""

import json
with open("alimentos.json") as file:
    datos=json.load(file)
print(datos)

('Alimentos': [({'Nombre': 'Manzana', 'Cantidad': 11}, {'Nombre': 'Pera', 'Cantidad': 22}, {'Nombre': 'Naranja', 'Cantidad': 33}]), ('Verduras': [({'Nombre': 'Lechuga', 'Cantidad': 88}, {'Nombre': 'Tomate', 'Cantidad': 55}, {'Nombre': 'Pepino', 'Cantidad': 66}]))
```

5.2 Expresiones regulares

Las expresiones regulares, llamadas también "regex", son patrones de texto que definen el formato que debe tener una cadena de texto.



Estos son algunos ejemplos:

^

El acento circunflejo o comienza en la cadena:
patron="^lo"

dichas cadenas pueden ser: "lopez", "lorena", "lobo"

\$

El dólar o termina en la cadena:

patron="on\$"

dichas cadenas pueden ser: "nación", "colaboración", "coronación", "pension"

^\$

El acento circunflejo y dólar coincide con la cadena exacta:

patron="^peru\$"

dicha cadena puede ser: "peru"

*

El asterisco es un cuantificador cero o más ocurrencia:

patron="a*b"

dicha cadena puede ser: "b","ab","aab","aaab"

+

El símbolo más es un cuantificador con uno o más ocurrencias:

patron="a+b"

dicha cadena puede ser: "ab","aab","aaab"

?

El signo de interrogación patrón opcional:

patron="a?b"

dicha cadena puede ser: "b","ab"

{}

El signo llave coincide exactamente con la ocurrencia del patrón:

patron="x{4}"

dicha cadena puede ser: "xxxx"

{,}

El signo llave coincide entre un rango con la ocurrencia del patrón:

patron="x{4,8}"

dicha cadena puede ser: "xxxx","xxxxx","xxxxxx","xxxxxxxx","xxxxxxxxx"

[]

El signo corchete clases de caracteres:

patron="amig[oa]"

dicha cadena puede ser: "amigo","amiga"

[-]

El signo corchete varios rangos:

patron="[0-9]"

dicha cadena puede ser: "0123456789"

patron=" [0-9A-Za-z]"

dicha cadena puede ser: "0Az"

|

El signo barra vertical permite definir opciones para el patrón:

patron="a|b"

dicha cadena puede ser: "a" o "b"

|?

El signo barra vertical permite definir opciones para el patrón:

patron="a|b|c?"

dicha cadena puede ser: "ac" o "bc" o "c"

()

El signo paréntesis agrupa patrones:

patron="(a|b)?"

dicha cadena puede ser: "ac" o "bc" o "c"

.

Puede ser cualquier carácter o es una clase predefinida

patron="s"

dicha cadena puede ser: "s","7","\s"

\	La barra diagonal invertida
\d	Coincide con cualquier dígito decimal [0-9]
\D	Coincide con cualquier carácter no dígito decimal [^0-9]
\s	Coincide con cualquier carácter que sea espacio en blanco [\t\n\r\f\v]
\S	Coincide con cualquier carácter que no sea espacio en blanco [^\t\n\r\f\v]
\w	Coincide con cualquier carácter alfanumérico [0-9A-Za-z]
\W	Coincide con cualquier carácter alfanumérico [^0-9A-Za-z]

A continuación, se muestran algunos métodos de las expresiones regulares:

match(): El cual determinada si la regex tiene coincidencias en el comienzo del texto.

search(): El cual escanea todo el texto buscando cualquier ubicación donde haya una coincidencia.

findall(): El cual encuentra todos los subtextos donde haya una coincidencia y devuelve estas coincidencias como una lista.

finditer(): El cual es similar al anterior, pero en lugar de devolver una lista, devuelve un iterador.

match.group(): Retorna la cadena encontrada.

match.span(): Retorna una tuple y contiene la posición (start, end).

match.start(): Retorna la posición inicial encontrada.

match.end(): Retorna la posición final encontrada.

5.2.1 Ejemplos

A continuación, se muestra un ejemplo del uso de formato de un número telefónico:

Expresion1.py

```
#programa : Expresion1.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
#-----#
descripcion : este programa muestra
el uso de Expresiones Regulares
validando un numero telefonico
#-----#
#importamos el modulo re de expresiones regulares
import re
#-----#
patron
telefono del Peru con el siguiente
```

Formato 00511 + Numero

00511 2645010

```

patron1=re.compile("00511\s[2-9][0-9]{6}")
#funcion de verificaciob de patron
def verificar(numero):
    if patron1.match(numero):
        return True
    return False

if __name__ == '__main__':
    while True:
        num=input("Ingrese Numero a Validar:")
        if verificar(num):
            print("Numero Correcto")
            break

```

Ingrese Numero a Validar:2645011

Ingrese Numero a Validar:00511 2645010

Número Correcto

A continuación, se muestra un ejemplo del uso de formato aplicado a una secuencia:

Expresion2.py

```

#programa : Expresion2.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de Expresiones Regulares
buscar un patron en una
secuencia de nucleótidos
"""

```

```
#importamos el modulo re de expresiones regulares
```

```
import re
```

```
#definir la secuencia a utilizar
```

```
secuencia="atcaatgatcaacgtaagcttctaaggcatgtcaagggtgctcacacagttagttccacaacctgagtggatg"
\      "acatcaagataggtcggttatccttcctcgactctcatgaccacggaaagatgatcaagagagaggatgattt"
\      "cttggccatatcgcaatgaatacttgtgacttgtgttccaaattgacatcttcagcgccatattgcgcgtggccaagg"
\      "tgacggagcgggattacgaaagcatgatcatggctgtttatctgtttgactgagacttgttaggata" \
"gacggttttcatcaactgactgccaaggcttacttgctgacatcgaccgtaaattgataatgaatttacatgc" \
"tcgttgc"
```

```
#definir una expresion regular
```

```
pattern = re.compile('[tc]a')
```

```
#busqueda segun patron
```

```
match = pattern.search(secuencia)
```

```
if match:
```

```
    print ("Patron encontrado :")
```

```
        +match.group())
```

```
    print ("Posicion Inicial,Final :")
```

```
        +str(match.span()))
```

```

print ("Posicion Inicial :"
+str(match.start()))
print ("Posicion Final :"
+str(match.end()))
else:
    print("No encontrado")

```

Patron encontrado :

Posicion Inicial,Final :(2, 4)

Posicion Inicial :2

Posicion Final :4

Expresion3.py

```

#programa : Expresion3.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de Expresiones Regulares
buscando ocurrencias
en un parrafo
"""
#importamos el modulo re de expresiones regulares
import re
parrafo="Buscar ocurrencias en este \
    "parrafo es importante para " \
    "demostrar el uso de findall" \
    "agradecere esta informacion" \
    "por que es vital paar el uso" \
    "de expresiones regulares"
#patron
patron1=re.compile("r[ae]")
#ocurrencias
ocurrencias = re.findall(patron1,parrafo)
#mostrar ocurrencias
print(ocurrencias)
print(vars(dict))
['re', 'ra', 'ra', 'ra', 'ra', 're', 're', 're', 're']

```

A continuación, se muestra un ejemplo de búsqueda de una ocurrencia específica:

Expresion4.py

```

#programa : Expresion4.py
#autor : jorge nolasco valenzuela
#fecha : 01-05-2020
"""
descripcion : este programa muestra
el uso de Expresiones Regulares
búsqueda de una ocurrencia especifica
"""

```

```
#importamos el modulo re de expresiones regulares
import re
#cadena
cad = "machu picchu es la octava maravilla del mundo Peru"
#Buscando si existe la ocurrencia
print(re.search("picchu",cad))
#hallar todas las palabras
print(re.findall("(.)u", cad))
#reemplazando
print(re.sub("mundo","world",cad))
<_sre.SRE_Match object; span=(6, 12), match='picchu'>
['hu', 'hu', 'mu', 'ru']
machu picchu es la octava maravilla del world Peru
```

Nota

Puede utilizar la siguiente URL para validar su expresión regular:

<https://regex101.com/>

The screenshot shows a web-based regular expression tester. The 'REGULAR EXPRESSION' field contains the pattern: / 00511\s[2-9][0-9]{6}/. The 'TEST STRING' field contains the value: 00511 2645012. The interface is clean with a light blue header and white background.

Preguntas

Capítulo 5



1. ¿Cuál es la diferencia entre clase y objeto?

2. ¿Existe alguna manera de verificar si una clase es subclase de otra?

- Sí
 No

3. ¿Para qué se usa la función Super()?

- Para hacer una mejor clase
 Para hacer una superclase
 Para acceder a los atributos y métodos de una superclase

4. ¿Qué suele ser una subclase?

- Más especializada que la superclase
 Una copia de la superclase
 Más general que la superclase

5. ¿Qué es un constructor?

6. ¿Qué es polimorfismo?

7. ¿Qué es herencia?

8. Crea la clase Película con sus respectivos atributos y métodos.

9. ¿Cuáles son los dos modos de acceso a archivos?

- () Binario y texto
- () Texto e imagen
- () Binario y ternario

Facebook:

<https://www.facebook.com/dogr4mcode.web>

Únete al grupo:

<https://www.facebook.com/groups/librosyrecursosdeprogramacion>

Referencias bibliográficas

Grandes proyectos hechos con Python (s. f.). Recuperado de <https://www.escuelapython.com/grandes-proyectos-hechos-python/>

Índice TIOBE para agosto de 2020 (s. f.). Recuperado de <https://www.tiobe.com/tiobe-index/>

PCAP Certified Associate in Python Programming certification (s. f.). Recuperado de <https://pythoninstitute.org/certification/pcap-certification-associate/>

PYPL Popularidad del lenguaje de programación (s. f.). Recuperado de <https://pypl.github.io/PYPL.html>

Referencias electrónicas

Django [Página web]

Disponible en <https://www.djangoproject.com/>

[Consulta: 21 de julio de 2020]

IronPython [Página web]

Disponible en <https://ironpython.net/>

[Consulta: 22 de julio de 2020]

Python [Página web]

Disponible en <https://www.python.org/>

[Consulta: 23 de julio de 2020]